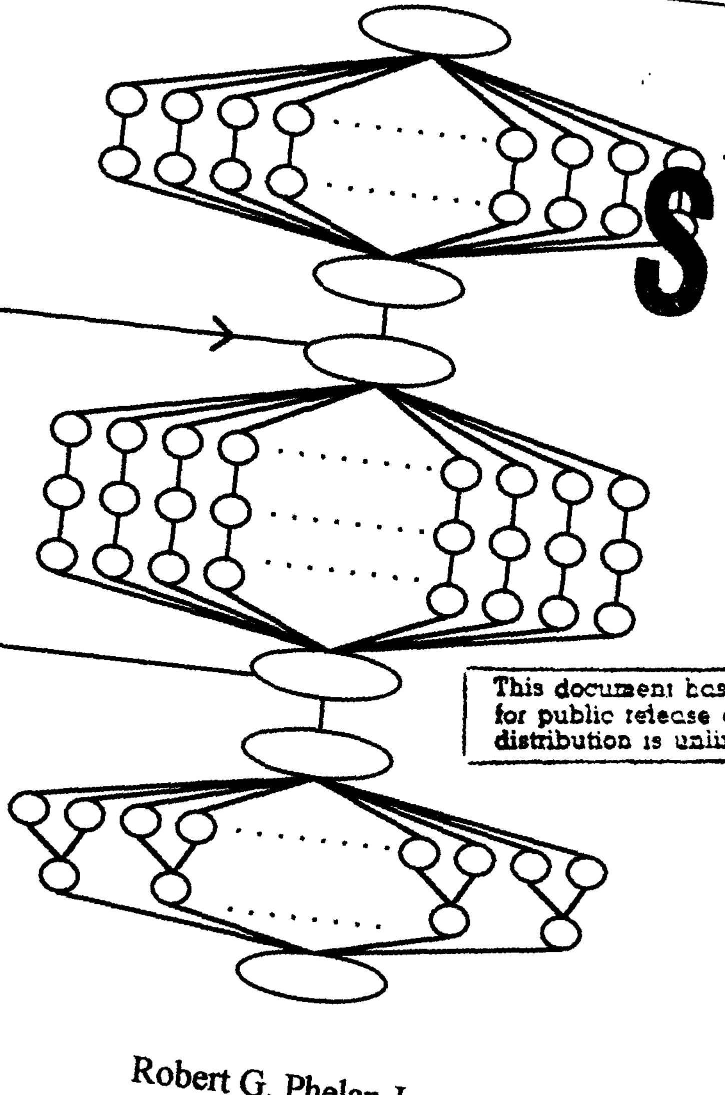
University of Virginia
Department of Systems Engineering

(1)

PARAMETRIC PARALLEL SIMULATION OF DISCRETE EVENT SYSTEMS ON SIMD SUPERCOMPUTERS



Robert G. Phelan Jr.

May 1994

PARAMETRIC PARALLEL SIMULATION OF DISCRETE EVENT SYSTEMS ON SIMD SUPERCOMPUTERS

A Thesis Presented to

The Faculty of the School of Engineering and Applied Science

University of Virginia

Department of Systems Engineering

In Partial Fulfillment

of the requirements for the Degree

Master of Science

in Systems Engineering

by

Robert G. Phelan Jr.

May 1994

Approval Sheet.

The thesis is submitted in partial fulfillment of the requirements for the degree of

Master of Science in Systems Engineering

[illegible]

Robert G. Phelan Jr.

This thesis has been read and approved by the examining committee:

[illegible]

Stephen G. Strickland
Thesis Advisor

Manuel D. Rossetti

Manuel D. Rossetti
Committee Chair

[illegible]

Andrew S. Grimshaw

Accepted for the School of Engineering and Applied Science:

Edgar A Starke, Jr.

Dean, School of Engineering
and Applied Science

May 1994

Acknowledgments.

- First and foremost, to my wife Kelly, for all her loving support and encouragement over the last two years with this transition from soldier to student. Without her, I could have never completed this thesis.

- To our children, Sean and Kaity, for always providing amusement, diversion, and unquestioning love whenever Daddy was home.

- Lastly, to Dr. Steve Strickland for the numerous hours we spent together. His immense patience and foresight allowed me to learn from my mistakes and still get this thesis completed on time, on target.

Abstract.

This thesis focuses on parametric parallel simulations using a Single Instruction Multiple Data (SIMD) supercomputer concentrating specifically on single server queueing systems. Extensions to networks of queues are also provided. In contrast to most previous work which distributed one simulation across processors, I simulate distinct parametric variations on each processor.

The SIMD architecture consists of a front end controlling computer and many parallel processors of simple design. The front end directs all subordinate processors to execute the same instruction at the same time. In performing SIMD simulations, a common event process is generated by the front end which is then used to update all simulations. I investigate two basic alternatives for generating this event process: time and event synchronous. The first, proposed by Vakili at Boston University, is called the Standard Clock Multiple System. This algorithm uniformizes the event rate across all states and parametric variants. This uniformized event process is generated by the front end and broadcast to the processors, where the events are thinned to local rates.

The event synchronous method generates a generic Poisson process on the front end. Each parallel processor then rescales time and types events based on local event rates and system states. All front end events result in local state updates as no thinning occurs with this approach.

These two basic approaches and several variants are compared with respect to event generation rates (wall clock time), measured performance variance, and covariance between successive variants. Methods are presented for predicting comparative algorithmic variance performance.

Table of Contents.

List of Tables.

List of Figures.

List of Symbols.

BE	Back End Computer Specific.
C	Variance Ratio Constant.
$E(x)$	Expected Value of x.
$f(t)$	Probability Density Function.
$F(t)$	Cumulative Density Function.
FE	Front End Computer Specific.
$h(t)$	Hazard Rate.
I	Binary (0,1) Indicator Variable.
k, k_{max}	Queue Buffer Sizes.
L	Queue Length.
N	Number of Processors.
$N(t), N'(t)$	Arrival or Counting Processes.
P, p	Probability.
P_{ij}	State Transition Probability.
q, q_{max}	Network Interevent Generation Rates.
R, r	Event Generation Rate.
r_i	External Arrivals to Network Queue i.
$\hat{R}_{\text{Time or Event Synchronous}}$	Event Generation Rates (System Time).
S	Speedup.
s	Number of Servers.
S^2	Sample Variance.
$S(t)$	State of System at Time T.
S_x, S_y	Computer Costs.
T, t	Time.

T(x)	Time to Execute x Simulations.
T_{Time}, T_{Event}, $T_{\mathrm{Time,Real}}$	Event Generation Rates (Real Time).
U	Uniform Random Number.
W	Queue Waiting Time.
$\bar{x}$	Mean of x.
α	Processor Normalization Constant..
$\alpha(t)$	Arrival or Counting Process.
β	Serial Component of Computer Code.
$\delta(t)$	Generalized Departure Rate.
ε, $\bar{\varepsilon}$	Efficiency, Average Efficiency.
$\bar{\varepsilon}_b$	Efficiency Crossover Point.
γ	Null Event Rate.
λ, λ_i, λ_{min}, λ_{max}, λ'	Arrival Rates.
μ, μ_i, μ_{min}, μ_{max}	Service Rates.
π	Limiting Probability.
θ	Error Constant.
ρ, ρ_i, ρ_{min}, ρ_{max}	Utilization Ratios.
σ_x, σ_x^2	Standard Deviation and Variance of x.
τ	Interevent time.
υ_i	Mean Time Spent in State i.

1.0 Introduction.

Parallel computers provide an efficient means with which to simulate a wide range of parametric variations of Discrete Event Systems in a short period of time. This thesis studies and compares the various methods available to simulate parametric families of systems of small to medium sized Markovian and Non-Markovian systems on Single Instruction Multiple Data (SIMD) supercomputers. The SIMD architecture consists of a front end controlling computer and many parallel processors of simple design. Two basic methods of event generation are introduced, discussed, and compared. The first method, *Time Synchronous,* generates local system events on the front end which are broadcast to the parallel processors such that the local system time is the same. The second method allows all systems to generate events based only on the current state of the system, so that all systems now operate at different system times and is called *Event Synchronous.* Selection of the proper event generation technique is based on some simple characteristics of the goals of the set of simulations. The question to keep in mind is whether a specified number of events are to be generated to compare all of the systems' performance or is the performance of the range of systems to be compared after a specified period of simulated time.

The primary difference between the two methods is that in the time synchronous method, not all events generated result in state updates. An event that does not result in a state update is essentially *inefficient.* That inefficient event wastes the processor's capabilities during the time that the other parallel processors are performing their local state updates. This inefficient use of the events results in a proportional rise in system performance measure variance as compared to the event synchronous method.

The time synchronous method can also be generated in two different ways. The first involves actual events generated by the controlling computer which are then broadcast to all subordinate processors for appropriate use. The second directs all subordinate processors to generate and type their own events. Selection of the particular time synchronous method directly affects the efficient use of the processors. Events are generated much more efficiently if they are typed on the local processors.

The computational time per event is different for the different methods. The time synchronous methods tend to generate events faster than the event synchronous. Comparison of the tradeoff between the real time to generate an event and the efficient use of the parallel processors is a major focus of this thesis.

1.1 Thesis Overview.

The remainder of Chapter 1.0 introduces the aifferent parallel computing hardware available. It also introduces the basic concepts of parallel simulation of *parametric variations of* Markovian *and* Non-Markovian *systems.* Lastly, it introduces the concepts of parallel simulation speedup and efficiency.

Chapter 2.0 defines the methodology used to setup parametric parallel simulations. It then develops the event and time synchronous methods of simulating Markovian queues. The probability formulations of the different methods are then described and compared. It lastly provides algorithms for implementation of these simulations methods on a SIMD parallel computer.

Chapter 3.0 defines processor efficiency for the time synchronous method. It analytically determines the efficient processor use of generated events for both M/M/1 and M/M/1/k parametric variations of queueing systems. It shows that inefficient events are generated as a result of local processor uniformization, as well as an attempted departure from state 0 or an attempted arrival from state k.

Chapter 4.0 describes how parametric variations are setup and broadcast to the parallel processors. It then provides a parallel algorithm for setting up the Alias Method.

It introduces a method with which parametric variation performance measures can be visualized across multiple processors. It shows how processor efficiency can be used to predict the time synchronous performance measure variance as compared to the event synchronous. It lastly provides experimental results for the M/M/1/k parametric family of queueing systems and compares the experimental with analytical re[illegible] It also compares the real time required to generate an event using any of the methods.

Chapter 5.0 extends the concepts developed for the single Markovian queueing systems to networks of queues.

Chapter 6.0 highlights the conclusions and contributions provided in this thesis. It also lists recommendations for future research.

1.2 Background.

Parallel computers have provided the scientific community with a means to solve large problems much more quickly than if they were performed on conventional serial computers. Consider the following example: Assume that a problem takes time T on a single processor to perform a single computer run. Then decompose the problem into N pieces to run simultaneously on N parallel processors. If the decomposition allows all processors to work continuously, then the time to execute the problem in parallel is T/N. Now suppose another parallel machine is available with 2N processors and the problem at hand can be decomposed into 2N pieces. Assume that these 2N processors are less expensive and run at a slower speed than the N processors in the first example. If the 2N processors run at a speed greater than or equal to 50% of the first N processors or have an unit cost of 50% or less than the N processors, then it would be more advantageous to use the second machine rather than the first. However, if no further decomposition can be performed or the second machine has no computational or cost advantages, then use the first machine.

The basic concept behind parallel computing is to take a big problem, break it up into N pieces, perform all tasks simultaneously on distinct processors, in one time interval, and reduce the elapsed computation time by a factor of N over the serial version assuming that equivalent processors and algorithms are used. Typically, there are some portions of any model/ algorithm that must be performed serially due to the interdependence of some calculations. Also, other portions of the algorithm may include instructions that must be executed by only one processor, such as opening a file for output. This *non-parallel* component tends to slow down overall parallel execution time. Each parallel processor is not utilized to its fullest capability. Many of them may be idle while portions of the serial component execute.

There are limits to the computing power of conventional computing. Serial computers consist of a control unit connecting the memory to a processor, and are commonly referred to as von Neumann machines. The control unit fetches instructions from memory one at a time and sends them to the processing unit for execution. Overall processing speed of the computer depends upon the rate at which instructions and data can be transferred between processor and memory. The connection between memory and the processor is commonly referred to as the von Neumann bottleneck.

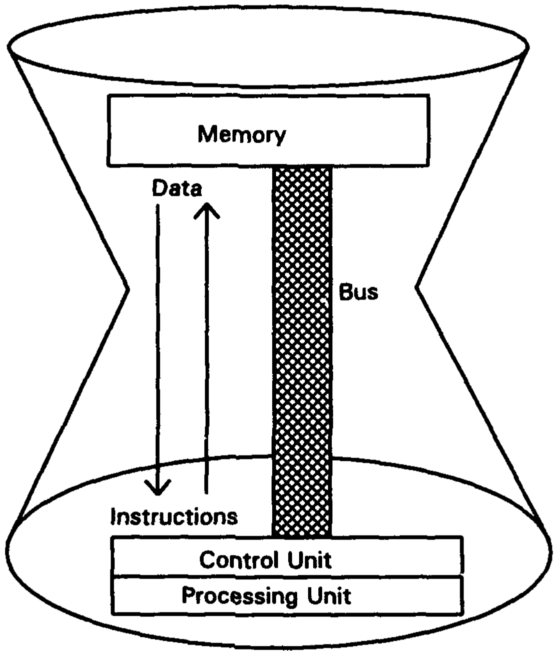


Figure 1.1 - von Neumann Bottleneck.[1]

The control unit fetches instructions from memory, which are then executed by the processor. The processor then requests either further instructions or data from memory. In this basic design, either instructions or data are using the bus at any one instant.

In the near future, the maximum switching speed of silicon chips will also be reached. A newer and most likely more expensive family of processors based on different materials, possibly gallium arsenide, will be developed to satisfy the never ending quest for computational speed. In the meantime, substantial increases in speed have become increasingly more difficult and much more expensive to achieve requiring large amounts of capital and substantial research.

[1] T.G. Lewis, H. El_Rewini, *Introduction to Parallel Computing*, Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1992, pg. 30.

There are many alternatives available to avoid the von Neumann bottleneck and gain computational speed. Pipelining instruction and data streams, multiple control units and use of alternative interconnections between the processor and memory are a few. These architectures are termed *non-von* because they do not follow the basic von Neumann design. Parallel computing architectures also fall into the non-von category.

Parallel computing seems to be one solution to both the approaching limit in silicon switch speed as well as the von Neumann bottleneck. Why design and build a computer around a faster processor when you could group slower and cheaper processors together? These processors could effectively perform the same computations at much less overall cost by decreasing the total time required to perform the task. By using available parallel computing technology, substantial capital investments could be amortized over longer periods of time and the pressure to dramatically improve computational speed would be somewhat relieved.

1.2.1 Flynn's Hardware Taxonomy.

The most widely used classification of parallel computational models was proposed by Flynn in 1972, (Figure 1.2), which classifies machines based on the nature of their instruction and data streams, each of which can be single (serial) or multiple (parallel). The von Neumann model is termed Single Instruction Single Data (SISD) within the framework of this taxonomy.

		Number of Data Streams	
		Single	Multiple
Number of Instruction Streams	Single	SISD (von Neumann)	SIMD* (vector, array)
	Multiple	MISD (feasible?)	MIMD (multiple micros)

Figure 1.2 - Flynn's Classification.[2]

SIMD * refers to Single Instruction Multiple Data machines which are also known as vector or array processors. The data is organized across the multiple parallel processors as arrays with the array indices corresponding to individual processors. MIMD refers to Multiple Instruction Multiple Data and one example can be envisioned as a group of microcomputers each performing individual tasks independently to complete a large computation task. There are also MIMD supercomputing platforms with substantial computational muscle.

1.2.2 SIMD Architecture.

SIMD machine processors simultaneously execute the same instruction sequence in lockstep, but on individual, local data elements. They require no external synchronization because each processor only accepts centrally issued commands for processing. No instructions are stored in local processor memory. Synchronization is effected by a central controlling processor, also known as a *Front End,* issuing

[2] G. S. Almasi, and A. Gottlieb, *Highly Parallel Computing*, The Benjamin/ Cummings Publishing Co., Redwood City, California, 1989, pg. 111.

instructions to all of the subordinate processors. Normally, SIMD architectures support many processors of simple design. Figure 1.3 depicts the logical layout of the SIMD architecture.

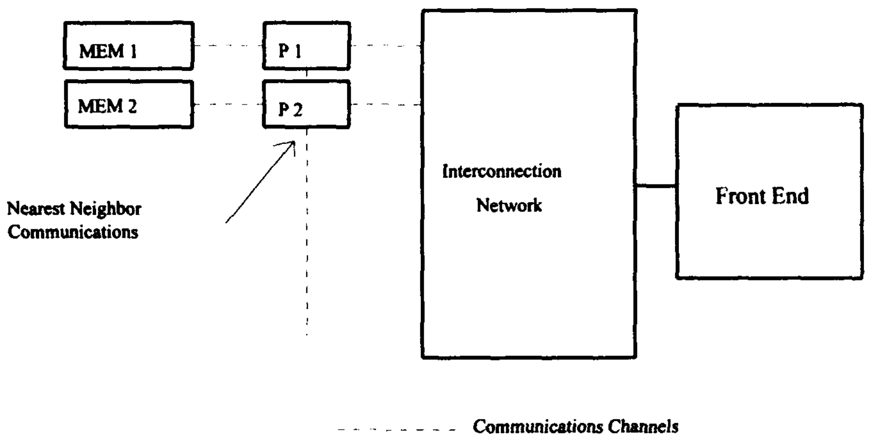


Figure 1.3 - Logical SIMD Architecture.

This type architecture is generally referred to as *Distributed Memory* as each individual processor has its own local memory which is usually very limited. If data located in memory x is required by processor y, it would be sent along one of two communications grids; either by the *Nearest Neighbor* network or through some interconnection network such as a boolean n-cube. Since the SIMD processors are of such simple design, more than one processor is normally physically located on a single hardware chip. The nearest neighbor network physically connects all *adjacent* processors located on a single chip. Chips are also physically connected to adjacent chips to facilitate nearest neighbor communications with adjacent processors located on adjacent chips. Figure 1.4 depicts a sample nearest neighbor physical layout. Not all interconnections are shown.

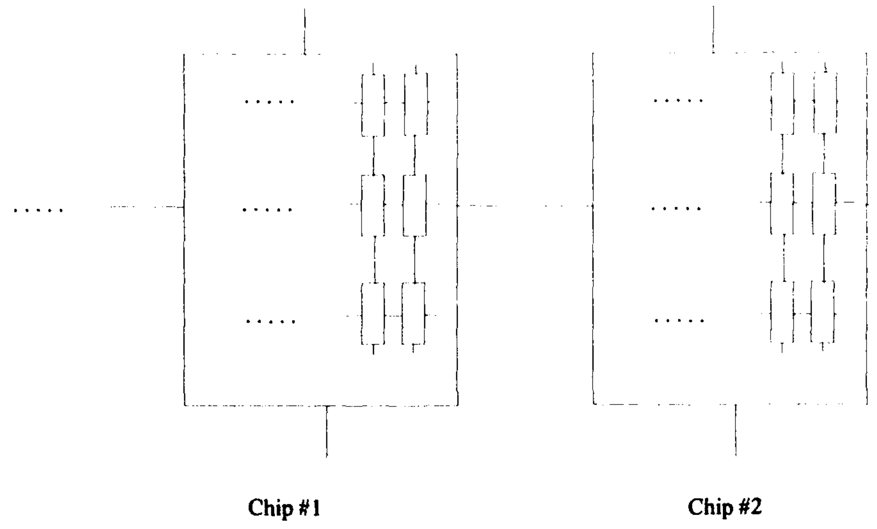


Figure 1.4 - General NEWS Network.

Data can be also exchanged between the front end and the local processors in any of three ways: broadcasting, global combining or a scalar memory bus. Broadcasting allows a single value from the front end to be replicated and sent to all processors at once. Global combining provides the front end with a means to obtain the sum, maximum, minimum, etc., of the *same* data element from each processor. The scalar memory bus, incorporated into the interconnection network, allows the front end to read or write to any individual processor when required.

1.2.3 MIMD Architectures.

Multiple Instruction Multiple Data (MIMD) architectures have been designed to allow individual parallel processors to operate independently of each other. One processor is usually designated as the 'master' while the remaining are termed 'slaves'. Processing is largely asynchronous and requires either the programmer and/ or the compiler to provide instructions to the master to provide the necessary synchronization and coordination of all slave processor activities.

MIMD *Shared Memory* architectures are also called 'dance hall' architectures. Essentially, all processors have the ability to read and write to all memory locations through the use of some communications interconnection network such as a bus or mesh. Figure 1.5 depicts a simple shared memory architecture.

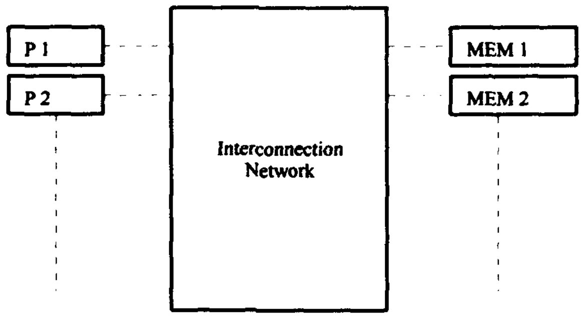


Figure 1.5 - Shared Memory Architecture.

MIMD *Distributed Memory* architecture is similar to the general SIMD architecture described in the previous section. A major difference is the fact that local memories now store instructions as well as data and the processors execute their instruction streams asynchronously. There is no front end but one of the processors has to perform overall synchronization to ensure that the application executes correctly. These machines are also called message passing machines because if they require an item stored in another processor's memory, they must send a message to request it and wait for the data to be returned. Message passing is another means to effect processor synchronization. Another way to visualize a distributed memory machine is to imagine a series of personal computers connected via ethernet with each computer running a portion of a large program. Figure 1.6 depicts a general distributed memory architecture. Any one of the processors P_i, could be the 'Master' in either of these configurations.

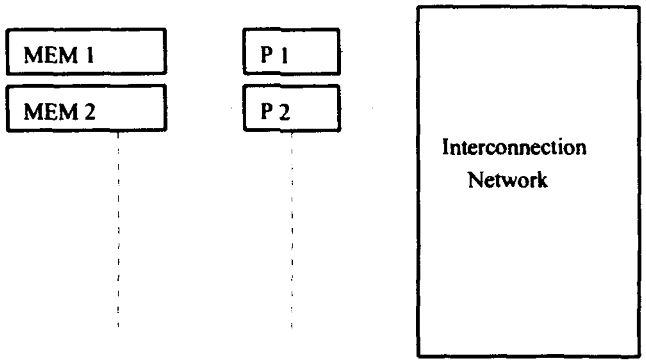


Figure 1.6 - Distributed Memory Architecture.

1.3 Problem Definition.

"A *system* is defined to be a collection of entities, e.g., people or machines, that interact together toward the accomplishment of some logical end"[3] Systems are also categorized as either discrete or continuous. The reason one performs simulations is to transform some real system into a model for analysis when an accurate analytical model can not be readily solved. Usually, some performance measure (*mean*) is measured and an estimate of how good that measure is (*variance*) is computed. Discrete-event simulation models a system as it progresses through time. A mathematical representation is used in which state variables change instantaneously in response to events occurring at distinct points in time. Typical events might be arriving at a checkout line in a supermarket, having the cashier begin to ring up the order and receiving your change thus ending your trip to the market.

[3] A. M. Law, and W. D. Kelton, *Simulation Modeling and Analysis*, 2nd Edition, McGraw-Hill, New York, 1991, pg. 3.

Considerable effort has gone into parallelizing 'large' system simulations by distributing portions of the simulation across multiple processors. There are different methods for decomposing simulations to execute upon the various hardware architectures of parallel computers. Parallel processors execute instructions either synchronously in the case of SIMD machines or asynchronously on MIMD machines. Intuitively, the more complicated are those which are to be executed asynchronously across multiple processors.

MIMD architecture suggests that a system must be decomposed for asynchronous computations. The first obvious difficulty encountered is how to break up the system. There are points within a simulation when the interdependence of computations require certain processors to wait for still others to finish before they are allowed to proceed with their individual instruction sets, e.g., computation of some performance measure. Any simulation decomposition must account for these requirements.

Efforts must be undertaken by the programmer and or the compiler to synchronize the overall efforts of individual processors. 'Message passing' implementations are synchronization schemes used on MIMD architectures for asynchronous distributed simulations. 'Time Warp' is another version of message passing. Consider the following Time Warp example: Processor 1 requests a data element from processor i. Processor 1's local system or simulation time is on the message. If processor i has a later system time, it's local system or simulation time essentially warps back to processor 1's time. If processor i's time is earlier, then the return message would provide a new local time to processor 1 who would then warp back to that new local time. Warping back in time requires discarding much, if not all, of the intervening computation that had been performed by the affected processor. Obviously, considerable synchronization is required and much computation can be wasted.

The SIMD architecture requires slightly different synchronization to effectively utilize the available processors and provide useful results. Recall that each processor is executing the same instruction at the same time. If an instruction does not pertain to a particular data set, then there is usually some sort of 'masking' at the individual processors that essentially shuts off the affected processors for that particular instruction sequence. Decomposition of a simulation would be extremely difficult especially in a complicated system. If the system is decomposed to run on a SIMD machine, then some processors would be idle as certain portions of the simulation progressed through the use of this masking feature. These processors would then become active during their portion and other processors would be masked out. For large decomposed systems, the available SIMD processors would then be drastically underutilized. SIMD programmers would like to efficiently utilize all of the available processors during the execution of their programs. It seems that simulations that have relatively little interdependence and whose representation can fit within a relatively small memory element would be well suited for SIMD machines.

There are many smaller systems that could be mapped to the SIMD architecture. Rather than attempt to decompose a large simulation, smaller systems could be modeled that fit within the limited memory of individual parallel processors. All SIMD processors would then simultaneously perform small simulations. An area of large interest is the effect of varying performance parameter values within a system to study the effect on the system's performance. The full range of systems to be studied can be considered a *parametric family*. SISD machines would require a simulation run at each parametric value of interest as well as repeated replications to compute any performance measures with reasonable variance. Imagine a family of systems in which there are y parametric performance factors to be varied over x different values. Conventional serial simulation techniques would require x^y independent runs, i.e., a run at each combination of possible values. Couple that with successive replications to compute an estimate of the expected value with reasonable variance, and the computational effort grows considerably. Assuming then that it takes one time interval to perform a single simulation, then to perform Nx^y replications of the run would take approximately Nx^y time intervals. SIMD clearly provides an architecture that allows each of the parametric variations coupled with multiple independent replications to be executed during one simulation run. Performing a wide range of parametric simulations provides an unique opportunity to understand the incremental effects of parameter variations that may not normally be observed using conventional techniques.

There are also some potential difficulties associated with any parallel architecture. The first is memory considerations. Since each individual processor has only limited local memory, the systems to be simulated have to modeled in such a way as to fit each parametric variation within available local memory. Also, there is a finite limit to the number of physical processors. The number of processors required grows exponentially as the number of different performance factors increases and geometrically as the number of range increments increases. Logically selecting subsets of all parametric variations, using some sort of experimental design technique, would help in reasonably maintaining the total number of simulations required.

Variance reduction techniques are another means to improve computational efficiency and speed. They should also be employed, if possible, to provide a reasonable estimate of the performance measure mean. There are many schemes with which to reduce variance. Inducing correlation among the observations seems particularly well suited to use by parallel computers through the implementation of common random number streams. There are different methods with which to simulate parametric families of systems by stochastically coupling them and inducing correlation to reduce the variance of the difference estimates. Rather than determining cardinal optimality, ordinal optimality of the parametric family of systems being simulated could then be determined faster, with more accuracy, and with subsequently much less computational effort.

1.3.1 Parallel Event Generation.

Discrete Event Simulation (DES) concerns the modeling of a system as it evolves over time by a representation in which the state variables change instantaneously at separate points in time. Consider the following generic DES algorithm:

INITIALIZI	Initialize Counters, Event List.
LOOP:	Pick next event off of list.
	Update state and time.
	Update performance measures.
	Schedule next event.
	Check stopping criteria;
	If met, go to STOP.
	Go to LOOP.
STOP:	Tally performance measures.

This algorithm provides an overview into the sequence of steps for a system simulation. During the INITIALIZE portion, all event counters are reset and the first possible events that can be scheduled are determined with a corresponding time increment. Most discrete event simulation models utilize a dynamic event list. This list maintains the set of possible transitions that the system can perform given that it is in its current state. As the simulation begins, the event with the lowest remaining time increment is picked and the system is advanced to that time. The simulation then checks to see which events need to be scheduled, if any, and schedules them as appropriate. The remaining events that are still valid then have their time increments adjusted by the amount of the event that was just picked. The simulation then proceeds to the next event, unless the simulation's stopping criteria is met. Another technique would be to use actual event occurrence times to avoid incrementally adjusting event schedule times. Stopping criteria could include either a specified stopping time or a number of generated events, based on the goals of the simulation.

There are several different methods for generating events in parallel. Vakili proposes a standard clock multiple system (SCMS) algorithm. "The single clock synchronizes all trajectories such that the 'same' event occurs at the 'same' time in all systems"[4] This algorithm is based on the well-known uniformization procedure for Markovian systems. Across multiple processors, it ensures that all parametric variations have the same simulated system time. Generating events in parallel requires procedures that ensure that all simulated systems use the next 'correct' sequential event. The standard clock technique creates a *master event sequence* on the front end which is broadcast to all subordinate processors., Local processors then dynamically use the master sequence to create their own sample paths.

During a discrete event simulation, system time *jumps* from one event to the next. The period between events is termed the interevent time, τ, and the assumption here is that no state changes have occurred to the system during that interevent time period. Figure 1.7 depicts a representative sample path of a discrete event system. The A's indicate arrivals and the D's indicate departures from this system. The τ_i's represent interevent times.

[4] P. Vakili, "Massively Parallel and Distributed Simulation of a Class of Discrete Event Systems: A Different Perspective", *ACM Transactions on Modeling and Computer Simulation*, 2, pg. 216.

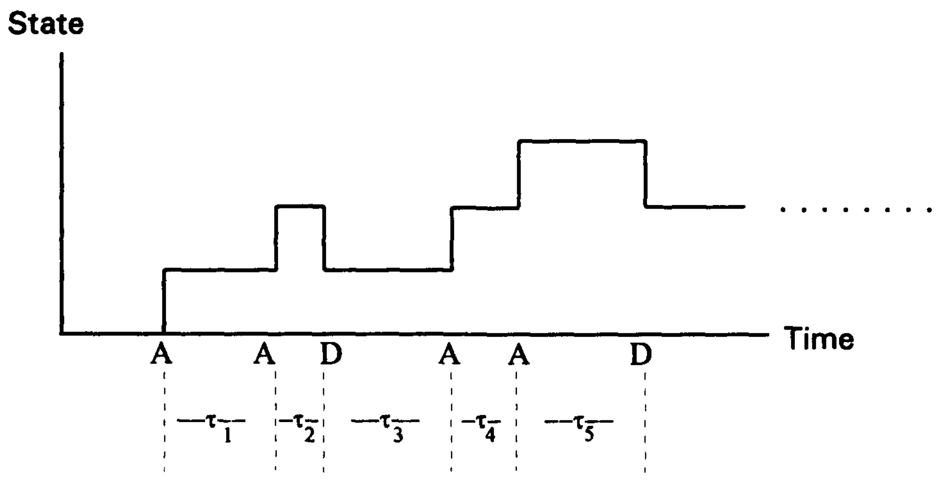


Figure 1.7 - State Diagram.

Discrete event parallel simulations should have events generated or system time incremented synchronously across all processors to ensure that parametric variations of the system in question can be more meaningfully compared. The standard clock increments time synchronously across all system variants.

1.3.1.1 The Poisson Process and Markov Chains.

A Poisson process of rate λ has exponentially distributed interevent times with rate λ. The definition of a Poisson process holds significant advantages for discrete event simulation because the exponential distribution has the *memoryless* property, which means that the process essentially probabilistically restarts itself from any point in time given that it has reached that instant in time with no events occurring in the interim. From a simulation point of view, the Poisson process then allows events to be scheduled dynamically or on the fly.

A Markov Chain is a stochastic process where the conditional distribution of any future state is dependent only on the current state and is independent of all past states.

Whenever the process is in state i, then a state transition to state j will occur with some probability P_{ij}. Simulating a Markov Chain then requires only the possible state transitions generated from the current state. In many simulation problems, the overall process might not be a true Markov Chain, but it could possibly contain an imbedded Markov Chain which would then determine a subset of possible state transitions. For an simple Markovian queueing system, possible events are customer arrivals and departures.

1.3.1.2 Poisson Splitting.

If a process of type I events is Poisson (λ), then interevent times of type I are exponential (λ). If a second independent process of type II event is Poisson (μ), then interevent times of type II are exponential (μ). If event type I occurs at an exponential rate λ and another independent event type II occurs at an exponential rate μ, then together they occur at an exponential rate ($\lambda+\mu$) and the combined or pooled process is then Poisson ($\lambda+\mu$). Taking the pooled Poisson process, we are now interested in splitting the process back into its two original processes. Assume that from the pooled process, events of type I are accepted with some probability p and those of type II with probability (1-p). This splitting of the process is depicted in Figure 1.8.

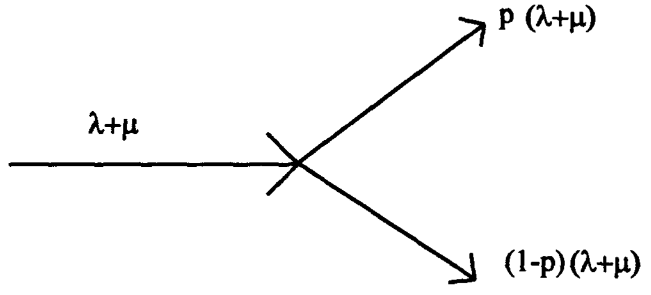


Figure 1.8 - Poisson Splitting.

It follows that if p $(\lambda + \mu)=\lambda$, then $p = \frac{\lambda}{\lambda + \mu}$. Also, if $(1\text{-}p)(\lambda + \mu)=\mu$, then $1 - p = \frac{\mu}{\lambda + \mu}$. For a complicated system, the pooled Poisson process can be split into as many subordinate processes as are occurring in the system. Figure 1.9 depicts this generalized splitting where λ denotes the overall pooled process.

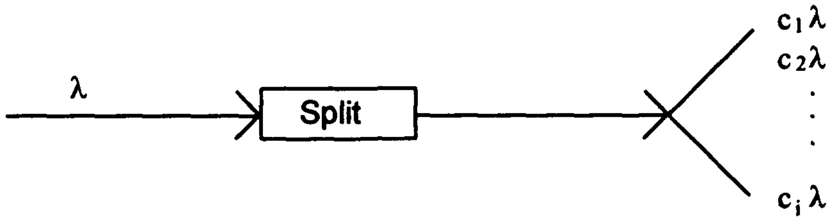


Figure 1.9 - General Poisson Splitting.

For the general split process, $\sum c_i = 1$. The resulting processes after the split could be considered marked processes, which means that they correspond to unique event types and are occurring at the correct original rate at which they were to be generated.

1.3.1.3 Poisson Thinning.

Poisson thinning is a variation of splitting. Suppose a process is Poisson (λ_i) of event type I. We wish to compare this process with another that is Poisson (λ_j) of event type I, where $\lambda_j > \lambda_i$. As stated before, a Poisson could be generated with rate $(\lambda_i+\lambda_j)$. However, now consider the process generated with rate λ_j. Events of rate λ_i could then be generated or thinned from the original process. Figure 1.10 depicts a 'thinning' of the event rate where $\lambda_i=p\lambda_j$, $p\leq 1$. Therefore, $p = \frac{\lambda_i}{\lambda_j}$ which is the probability of accepting the lower rate process as generated from the higher. Generating events at rate λ_j and accepting them with probability p will generate the λ_i process.

$$\begin{tikzcd} {} \arrow[r, "\lambda_j"] & \boxed{\text{Thin}} \arrow[r] & p\lambda_j = \lambda_i \end{tikzcd}$$

Figure 1.10 - Event Rate Thinning.

If the input process is a marked process, then the resulting output process would also be considered marked because the events are still of the same type, I.

The Standard Clock algorithm generates a marked Poisson process for use by all parallel processors. A pooled Poisson process is split to its maximum component event rates. Each parallel processor must then thin the marked event to its rate of interest. There are also many event generation schemes with which these Markovian systems can be simulated across parallel processors.

1.3.1.4 Non Homogeneous Poisson Thinning/ Splitting.

Consider now a non homogeneous Poisson process that has event rate $\lambda(t)$, (Figure 1.11). Note that $\lambda(t)$ could also be a continuous function. Essentially the event rate is a function of the process time. The non homogeneous process can be uniformized to a homogeneous Poisson process of rate λ by the following algorithm:

```
1) Let $\lambda_{max} = Max_t\{\lambda(t)\} \forall\ t \leq T$.
2) Generate an interevent Poisson($\lambda$).
3) 'Count' events with probability $\lambda(t)/\lambda_{max}$ at time t.
```

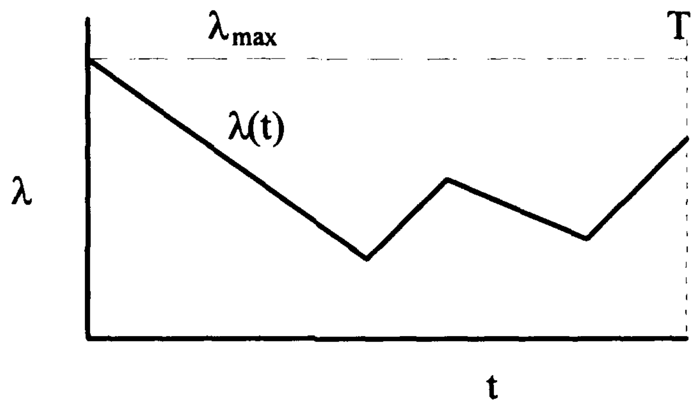


Figure 1.11 - Poisson Scaling.

The ratio of the heights, $\frac{\lambda(t)}{\lambda_{max}}$ is the probability of acceptance of the event.

1.3.1.5 *Parallel Poisson Splitting/ Thinning.*

The process of Poisson splitting can be applied across multiple processors to simulate parametric families of systems. Assume for this example that we are interested in simulating a Poisson parametric family of systems by varying the rate of events of type I (λ) and II (μ). Further assume that λ will be held constant across all of the processors. Let $\mu_i = \mu$ at the ith processor, $i = 1..N$, and let $\mu_{max} = MAX(\mu_i)\forall i$. Further assume that the required interevent generation locally at each processor is at rate Poisson ($\lambda+\mu_i$). Events of type I are then marked according to the following probability formulation :

$$P(\text{event type I}) = \frac{\lambda}{\lambda + \mu_{max}} \tag{1.1}$$

All processors would accept and generate this event because no thinning is required. Event type II requires both a mark (split) and then thinning to generate events at the proper rate.

$$P(\text{event type II}) = \underbrace{\frac{\mu_{max}}{\lambda+\mu_{max}}}_{\text{(mark)}} \underbrace{\frac{\mu_i}{\mu_{max}}}_{\text{(thin)}} = \frac{\mu_i}{\lambda + \mu_{max}} \qquad (1.2)$$

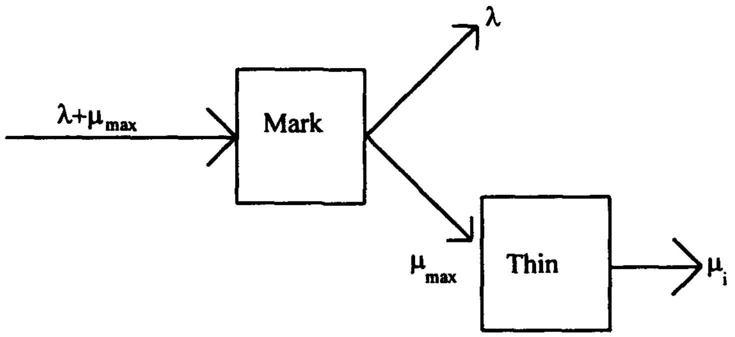


Figure 1.12 - Parallel Poisson Marking/ Thinning.

Splitting the process using this technique marks the resulting event into either type I or II with the appropriate event rate. Extending the Poisson marking/ thinning process to parallel processors is the main idea behind Vakili's 'Standard Clock' algorithm. Interevent times are synchronized across all systems generated in parallel. Poisson events are uniformized and generated at the fastest rate of all parametric variations of the system in question. The events are then 'thinned' at each individual processor. System time can be maintained in one central memory location.

Assume for the following example that we are once again interested in simulating a Poisson parametric family of systems by varying the rate of events of type II (μ). Let μ_{max} = Max(μ_i), i=1..N. The interevent rate is Poisson($\lambda+\mu_{max}$) as in the previous example. Events are thinned and essentially thrown away by the ratio of the local event type II rate to the maximum event type II rate. If the resulting process is provided to the local processors as **unmarked**, it then requires splitting and thinning at the local processor to identify the specific event type at the proper rate. The underlying Poisson process at processor i then becomes: Poisson($\lambda + \mu_{max}$) $\Rightarrow$ Poisson($\lambda + \mu_i + \gamma$). Event rate γ is essentially a 'null' rate and can be thought of as an event type II that would have occurred from the maximal rate process but does not occur at processor i. This thinning/ splitting can be incorporated into one set of probability equations, given that unmarked events are generated at the maximal rate on the front end, and marked at processor i for the selection of the event type:

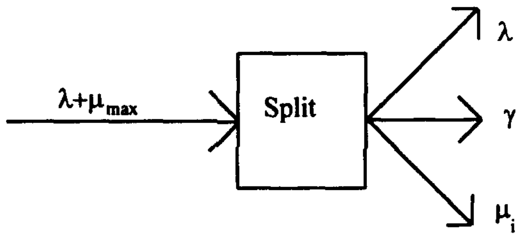


Figure 1.13 - Parallel Poisson Thinning/ Splitting.

$$\text{P(event type I)} = \frac{\lambda}{\lambda + \mu_{max}} \tag{1.3}$$

$$\text{P(event type II)} = \frac{\mu_{max}}{\lambda + \mu_{max}} * \frac{\mu_i}{\mu_{max}} = \frac{\mu_i}{\lambda + \mu_{max}} \tag{1.4}$$

$$\text{P('Null' event type)} = \frac{\mu_{max}}{\lambda + \mu_{max}} *(1 - \frac{\mu_i}{\mu_{max}}) = \frac{\mu_{max} - \mu_i}{\lambda + \mu_{max}} \tag{1.5}$$

Figure 1.14 illustrates the thinned or uniformized Markov chain for the process described by equations (1.3) - (1.5) for states j..j+2 at processor i.

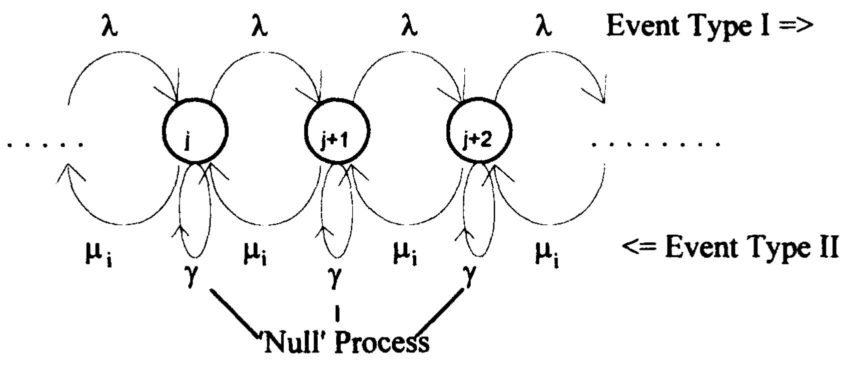


Figure 1.14 - Thinned Markov Chain.

1.3.1.6 Non- Exponential Distributions.

Poisson scaling also can be extended to use by parallel processors. Parallel thinning and splitting techniques have to be employed concurrently to ensure process synchronization. Once again, Let $\mu_i = \mu$ at the ith processor, $i = 1..N$ and Let $\mu_{max} = \text{Max}(\mu_i)$, $i = 1..N$. Events are then generated at each processor at the rate Poisson($\lambda+\mu_i$).

Using the basic parallel splitting scheme, let $\lambda_{max} = \text{Max}_t\{\lambda(t)\}\forall\, t \leq T$. Events are once again marked before they are sent to the individual processors. Equations 1.1 and 1.2 then become:

$$P(\text{event type I}) = \frac{\lambda_{max}}{\lambda_{max} + \mu_{max}} * \frac{\lambda(t)}{\lambda_{max}} = \frac{\lambda(t)}{\lambda_{max} + \mu_{max}} \qquad (1.6)$$

$$P(\text{event type II}) = \frac{\mu_i}{\lambda_{max} + \mu_{max}} \qquad (1.7)$$

Consider the same process but now, events are unmarked as they are sent to the parallel processors. Again, let $\mu_{max} = \text{Max}(\mu_i)$, $i=1..N$, and $\lambda_{max} = \text{Max}_t\{\lambda(t)\}\forall\, t \leq T$.

The rate at which interevents would be generated is Poisson (λ_{max}+μ_{max}). The following equations both mark and thin the process.

$$P(\text{event type I}) = \frac{\lambda_{max}}{\lambda_{max} + \mu_{max}} * \frac{\lambda(t)}{\lambda_{max}} = \frac{\lambda(t)}{\lambda_{max} + \mu_{max}} \tag{1.8}$$

$$P(\text{event type II}) = \frac{\mu_i}{\lambda_{max} + \mu_{max}} \tag{1.9}$$

$$P(\text{'Null' event type}) = \frac{\mu_{max} - \mu_i}{\lambda_{max} + \mu_{max}} + \frac{\lambda_{max}}{\lambda_{max} + \mu_{max}} * (1 - \frac{\lambda(t)}{\lambda_{max}})$$

$$P(\text{'Null' event type}) = \frac{\mu_{max} - \mu_i + \lambda_{max} - \lambda(t)}{\lambda_{max} + \mu_{max}} \tag{1.10}$$

Simulating a non-exponential distribution require the use of its hazard rate because the distribution is no longer memoryless. The hazard rate is interpreted as the conditional probability density that the event will occur in the next dt time units given that it has not occurred up until time t.

$$h(t) = \frac{f(t)}{1 - F(t)} dt \tag{1.11}$$

Figure 1.15 illustrates the hazard rate, h(t), density, f(t), and distribution, F(t), functions for the Weibull(3,1) distribution. In a similar manner to the non homogeneous Poisson process defined earlier, replace $\lambda(t)$ with h(t), and λ_{max} with h_{max}. Events would then be accepted with rate $\frac{h(t)}{h_{max}}$.

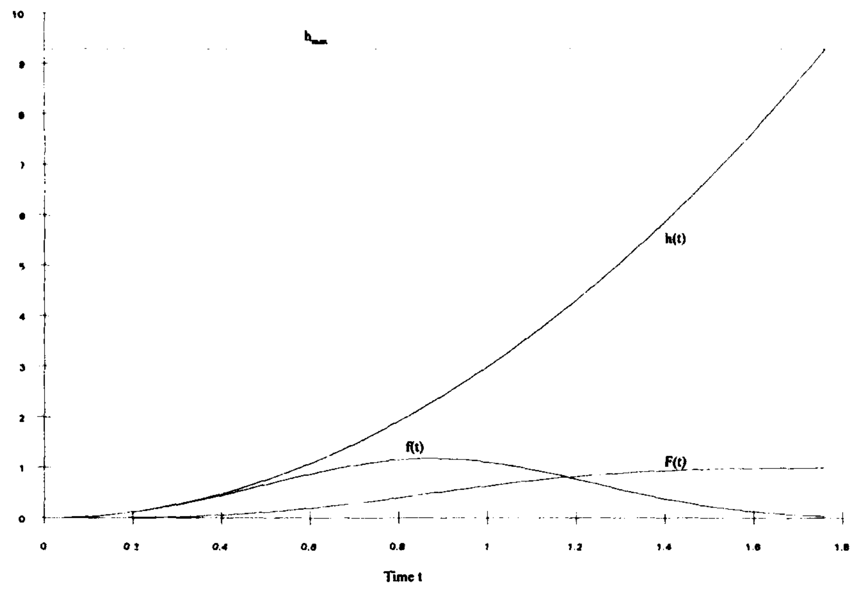


Figure 1.15 - Unbounded Hazard Rate.

The h_{max} for this particular distribution was selected as $F(t) \approx 0.995$ for illustrative purposes. The following formulation provides the basis for parallel event generation of non exponential distributions using the relationships developed in Equations 1.8-1.10 for an unmarked process.

$$P(\text{event type I}) = \frac{\lambda(t)}{\lambda_{max} + \mu_{max}} = \frac{(\frac{f(t)}{1-F(t)})dt}{h_{max} + \mu_{max}} \tag{1.12}$$

$$P(\text{event type II}) = \frac{\mu_i}{h_{max} + \mu_{max}} \tag{1.13}$$

$$P(\text{'Null' event type}) = \frac{\mu_{max} - \mu_i + h_{max} - (\frac{f(t)}{1-F(t)})dt}{h_{max} + \mu_{max}} \tag{1.14}$$

This technique will not work when the distribution's hazard rate is unbounded.

1.3.2 Hardware Selection.

Essential to parallel computation is the selection and use of the correct hardware for the particular application. Obviously, certain machine architectures are better suited to different applications. The primary difference between the hardware configurations is whether computations are to occur across large sets of data and require very little interaction or would benefit from different processors acting independently on different portions for periods of time then performing any inter-processor communications. The SIMD architecture seems better suited to the study of parametric parallel simulations because of the requirement to perform many simulations independently with the only processor interdependence occurring during the computation of performance measures.

The basic concept of the SIMD machine requires a front end computer capable of controlling thousands of individual processors of much simpler design. The front end computer then will cost substantially more than the per-processor cost for each individual parallel processor and will probably run at a much faster processing speed as compared with an individual processor. However, the benefits associated with the use of many parallel processors outweigh the cost of one front end computer as compared to a MIMD machine which would have substantially fewer parallel processors with greater computational capabilities and speed .

Assume that X parallel processors in a MIMD architecture cost $\$S_x$. Further assume that the front end of a SIMD machine is a more expensive processor than that used in the MIMD machine and costs $\$S_y$. These costs include any memory costs. The total cost of the SIMD would be S_y+NY, where Y is the per processor cost of the SIMD parallel processors. The MIMD machine would then cost XS_x. Consider then that there exists an obvious tradeoff between speed and cost. For the purposes of this discussion, define α as a normalization constant that allows performance comparison of the SIMD and MIMD parallel processors. Define $\bar{\varepsilon}$ as the effective utilization rate of the SIMD parallel processors for the application in question. For the purposes of this simple comparison, assume that the MIMD uses its processors 100% of the time. What then is the relationship between α and N? By attempting to equate the two architectures, $N = \frac{\alpha X S_x \bar{\varepsilon}}{Y} - S_y$. If α=10, then the MIMD processors operate ten times as fast as their SIMD counterparts. Let $\bar{\varepsilon}$=0.9, then 90% of the SIMD processors are used continuously throughout the course of the application. If X (# of MIMD processors)=64, S_x= \$2,000, S_y= \$4,000 and Y= \$20, then N= 53,600 processors. Given these very crude cost assumptions, it becomes obvious that the SIMD architecture can be cost effective if the application to be mapped to it is inherently parallel. The loss in speed is more than made up for in the sheer volume of computation. Any problem requiring largely asynchronous operations, would be much better suited to a MIMD machine.

1.3.2.1 Hardware.

The primary computer that will be used in conjunction with this research is a *Connection Machine-2* (CM-2) manufactured by Thinking Machines, Inc. This machine is a large SIMD machine and is physically located at the Pittsburgh Supercomputing Center (PSC). The principle advantage in using SIMD computers is an architecture that readily exploits the data parallelism inherent in the simulation of parametric families of systems and the similarity of instruction sequences. There is very little need for inter-processor communication during most of the execution of a simulation but some communication is required during the wrap-up phase when performance measures are computed.

The CM-2 used for this research is driven by a SUN 4-470 'front-end' computer. This front end is responsible for compilation of CM applications programs and then transferring instructions and data to the parallel processors of the CM-2. It also performs calculations and stores global variables in front-end memory.

The PSC CM-2 contains 32,768 bit serial processors. Bit serial processors, as implied by their name, only act on one bit of data at any time. The front-end issues instructions to allow these bit-serial processors to perform operations on larger data structures. The CM-2 also contains 1,024 64-bit floating point accelerator chips. The CM-2 processors are physically grouped 16 processors to a single chip. CM-2 processor chips are arranged in pairs and share a group of memory chips, a floating point accelerator and a floating point execution chip.

The PSC CM-2 has been partitioned into one 16K and two 8K regions so more users can be serviced at any one time. Under timesharing, a user normally attaches to one of the 8K partitions. For large jobs, a batch file scheduler is available to utilize either 8, 16 or 32K of the processors

CM-2 inter-processor communications utilize a 12-dimensional hypercube implemented on top of a 16 dimensional Cartesian grid. It is possible to employ one-dimensional to sixteen-dimensional nearest-neighbor grid communications according to the requirements of the task. This nearest neighbor communications with up to 16 other processors is commonly referred to as *NEWS* (North, East, West, South) for ease of understanding. The *NEWS* grid allows communications to occur between adjacent processors in one communication hop along the Cartesian grid. The nearest neighbor grid is etched directly onto the processor chips. During execution, some individual processors may have to communicate with a neighbor that is not physically located on the same chip. This communication is performed on a specialized independent per-chip permutation circuit outside of the 12 dimensional hypercube wiring. Instructions from the front end and general non nearest neighbor inter-processor communications occur over the hypercube. Figure 1.16 shows the interconnections for one processor chip and its associated memory for a Connection Machine.

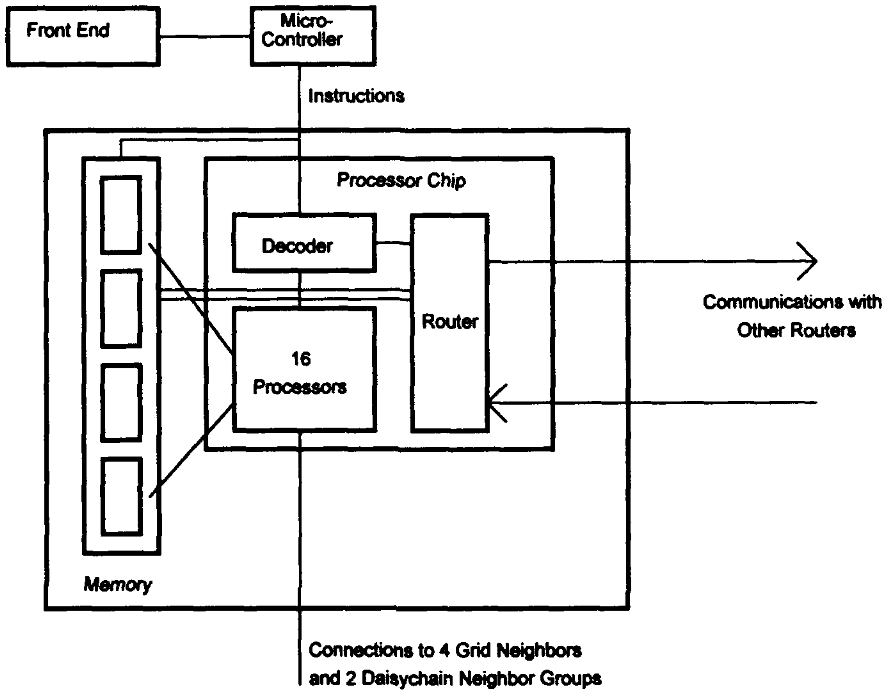

Front End
Micro-Controller
Instructions
Processor Chip
Decoder
Router
16 Processors
Memory
Communications with Other Routers
Connections to 4 Grid Neighbors and 2 Daisychain Neighbor Groups

Figure 1.16 - Connection Machine Processor and Memory Layout.[5]

1.3.2.2 Software.

The two compilation models available are Paris and Slicewise. Under the Paris model, the basic processing elements are the bit-serial processors with 8K of local memory. Under the Slicewise model, the CM-2 is organized into processing nodes, each containing 32 bit-serial processors, 256K of local memory and one floating point accelerator chip. Slicewise uses 32 processors together to do a single 32-bit operation in 1 clock cycle as opposed to Paris doing 32, 32-bit operations in 32 clock cycles. When a

[5] G. S. Almasi, and A. Gottlieb, *Highly Parallel Computing*, The Benjamin/ Cummings Publishing Co., Redwood City, California, 1989, pg. 335.

program is run under the Paris model at PSC, there are 32K processing elements available whereas under the Slicewise, only 1K. The Slicewise model makes use of the floating point accelerator's internal registers to perform computations. In order for the Paris model to perform any computations, each processor must serially read and write each bit.

For problems that require more than the available number of processors, the CM-2 uses the notion of virtual processors to simulate a large number of processing elements by allocating more than one virtual processor to each physical processor. Each processing element executes its instructions as many times as there are virtual processors assigned. This ratio or number of required loops is also referred to as a VP ratio.

Since the physical memory is divided among the virtual processors, the amount of memory at each processing node will be the available physical memory divided by the number of virtual processors in use at that processor. Under Slicewise, with a virtual processor ratio of 2, each processing element will have 128K of local memory. Under Paris, the same processing element will have 4K.

Both compilers allow the programmer to logically map the virtual processors if any inter-processor communication is known prior to execution. Data can be placed locally on each processing element if it is known that computations will be performed between two data elements of a serial array. The dimensions of an array can also be logically aligned to optimize any inter-processor communications. If NEWS is indicated, then the processors along a particular axis are to be physically connected if their grid indices differ by 1. If it is known that communications will occur along a particular axis but not necessarily between adjacent neighbors, the compiler allows these processors to be logically connected to minimize communication time along the hypercube.

1.3.3 System Variants.

One of the major goals of Discrete Event Simulation is to produce estimates of the means and variances of system performance measures. A significant advantage of a CM-2 computer is in its ability to generate a large number of system variants. Why not then logically set up the parallel computer to generate both a set of system variants as well as multiple replications of the variants different random number streams? The mean and variance of the performance measure in question can then be computed during a single execution of the simulation. Figure 1.17 depicts a processor layout in which the system in question has two parametric values varied along the x and y axes with the random number streams generated in the planes formed by the z axis. Identifying any individual processor is equivalent to identifying a point within a cube by its coordinates: (x,y,z). All processors with the same z coordinate simulate parametric variants. Stochastic variants are then identified by keeping x and y constant and varying the z coordinate.

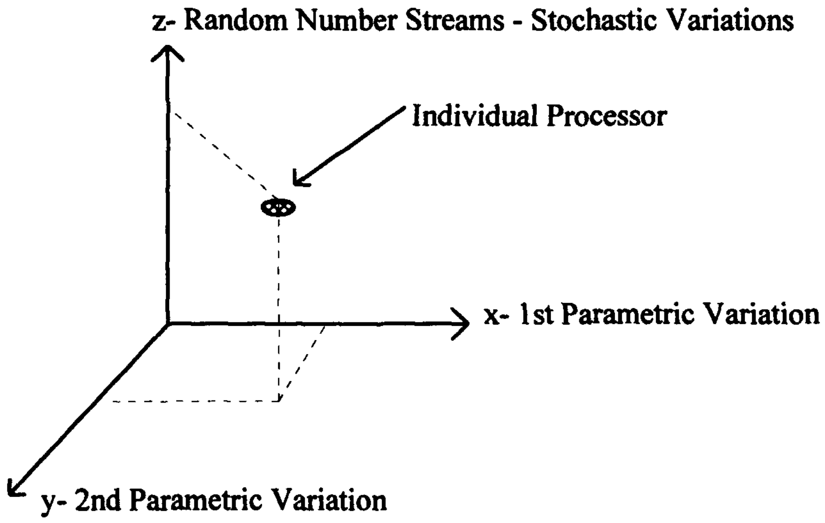

z- Random Number Streams - Stochastic Variations
Individual Processor
x- 1st Parametric Variation
y- 2nd Parametric Variation

Figure 1.17 - Processor Organization.

The mapping illustrated in Figure 1.17 could easily be extended to additional dimensions if more parameters are to be varied during the course of a simulation execution. One possible implementation of this three dimensional mapping is the M/M/1/k queue where the arrival rate would be held constant, the service rates varied along the x-axis, and the buffer size varied along the y-axis.

The number of stochastic variants formed by the z-axis should be at least 30 as a rule of thumb. The selection of 30 random number streams allows the value of the sample variance, $E(S^2) \cong \sigma^2$, the true variance. Normality assumptions which underlie most statistical analyses should be reasonable.

1.4 Performance Estimation.

Parallel simulation of parametric families of systems can provide clear insight into the performance of any system over a very wide range of possible system configurations. Comparison of critical performance measures is crucial to the success of the parallel simulation run. An obvious difficulty is the effective representation of multi-dimensional data. Even after the data is generated, there is the obvious question of how to interpret the results and provide meaningful insight into the system's overall performance.

Critical to the success of any simulation is the proper identification of its goals and which performance measures should be evaluated. Is the system in question to be optimized to provide maximum use of its resources or is the system being compared to another system of different performance parameter values? There are four possible general classifications in which simulation performance estimates can be grouped. [BFS 1987].

	Finite Horizon	Infinite Horizon
Absolute Performance	I	II
Relative Performance	III	IV

Figure 1.18 - Performance Measure Differences.

Finite Horizon refers to a specific time when a system reaches a point when a simulation is stopped to measure a performance measure. Infinite horizon refers to steady state conditions which a system may or may not ever reach. Absolute performance refers to the computation of the actual performance measure for that particular horizon as compared to relative performance which provides an ordinal ranking.

Evaluation of the performance measure's response is only valid if the simulation provides an accurate estimate as to its true value. Several factors influence the *goodness* of the estimate. Another important aspect of any simulation is the reduction of the performance measure variance to within acceptable limits so as to provide an accurate measurement of the performance measure's estimate. Based on the horizon and performance measurement requirements, different techniques can be used to achieve a suitable variance and resulting confidence interval.

Infinite Horizon: Increase the simulation time to infinity until a steady state solution is reached, if any.

Finite Horizon: Increase the number of independent replications of the simulation.

Relative Performance: Correlate the performance measure by using common random numbers.

$$\mathrm{Var}(X-Y)=\mathrm{Var}(X)+\mathrm{Var}(Y)-2\mathrm{Cov}(X,Y) \qquad (1.15)$$

As the covariance increases, the resulting variance of the differences decreases.

Another useful statistical measurement is the *Coefficient of Variation*, a dimensionless quantity that measures the amount of variability relative to the value of the performance measure's mean. $\text{C.V.} = \frac{\sigma}{\bar{x}}$. Given the possible wide range of performance measures that could be generated on the CM-2, this measure should provide a more accurate representation of the comparative overall performance of the simulations. It combines both standard deviation and mean data, thereby possibly reducing data interpretation requirements.

1.5 Speedup and Efficiency.

The most important and controversial aspect of the performance of a parallel computing application is the speedup. There are widely diverse schools of thought on the maximum speedup possible. Let N be the number of processors. Then the speedup, S, for parallel simulations, is defined as the following:

$$S = \frac{T(1)}{T(N)} \equiv \frac{\text{Time to execute M simulations on one processor}}{\text{Time to execute M simulations on N processors}} \tag{1.16}$$

Two laws have been developed to predict the potential speedup of any application. Key to both is a measurement of the inherently serial portion, β, of the code. This measurement is done by profiling the code and timing relative portions and determining the percentage of the serial portion to the total.

Amdahl's Law assumes a fixed amount of computation, and computes the speedup as the ratio of the processing time on N processors to that of a single processor. Assume that one simulation takes T seconds and M simulations take MT seconds. The serial part of the program can be computed in time βT(1) and the parallel part in (1-β)T(1)/N. The time required for M simulations across N processors would then be NβMT + (1-β)MT seconds.

$$S = \frac{MT}{\beta NMT + (1-\beta)MT} = \frac{1}{\beta N - \beta + 1} = \frac{1}{\beta + \frac{1-\beta}{N}} \tag{1.17}$$

As the number of processors increase, the achievable speedup tends to 1/β. This law then implicitly assumes that β and N are independent of one another.

Derivation of the Gustafson-Barsis Law assumes that the serial fraction and the number of processors are dependent on each other. As the number of processors increase, the serial portion shrinks proportionately. For the same M simulations previously used in the Amdahl's law example, if only one processor is used, T(1) =MN(1-β)T+βMT, since the one processor will execute both portions of the code. Now scaling up the problem to N processors only requires dividing the serial portion of the code by N by assuming that it also can be *amortized* across the processors; T(N)=βMT+(1-β)MT.

$$S = \frac{MN(1-\beta)T + \beta MT}{M\beta T + (1-\beta)MT} = \beta + N(1-\beta) \tag{1.18}$$

Note that β is no longer the fraction of serial code, but rather the fraction of serial code in a problem of size 1. As the number of processors increases, speedup also increases because the serial component remains relatively fixed. To compare the two laws and determine their applicability to SIMD machines, let N=1000 and β=0.10. Amhdahl predicts a speedup of 9.9 whereas Gustafson-Barsis predicts 900.1. Given the inherent parallelism of the SIMD architecture, coupled with the relative growth of the parallel portion of the parametric simulation code as the number of processors increases, predicted speedup is governed by Gustafson-Barsis.

The predicted speedup assumes that all processors are used 100% effectively during the parallel portion of the simulations. However, Poisson splitting and thinning 'throw out' events. For that particular event generation phase of the simulation, some individual processors remain *idle*. The state of the local system is not changed and the local system time is updated at the front end for front end marked events. Processors that are idle would require generation of *extra* events to bring their total event count up to the maximal rate system. The overall idleness of the parallel processors then has to be factored into the parallel portion of the Gustafson-Barsis speedup equation to provide an accurate representation of the achieved speedup. System parametric variations should be selected in order to minimize the number of processors that remain idle but still perform the necessary simulations. Equation 1.18 then becomes:

$$S = \beta + N(1-\beta)\bar{\varepsilon} \tag{1.19}$$

$\bar{\varepsilon}$ is defined as the average overall efficiency of the processors during the parallel portion of the simulations.

2.0 Markovian Queueing Systems.

Chapter 2.0 defines the basics of queueing theory and applies these basics to massively parallel simulations. The Event and Time synchronous event generation methods are fully developed and implementation algorithms are provided.

2.0.1 Background.

A *Queue* or *waiting-line* system is a facility or group of facilities maintained to meet the demand for some service by a population of individuals or units.[6] Some commodity flows through the queue in order to receive the provided *service.* Queueing theory provides a large number of alternative mathematical models for describing a waiting-line situation. The basic process used by most queueing models is as follows. *Customers* requiring service are generated by an *input source* from some customer population which can be either finite or infinite. These customers then enter the queue and wait for their turn in service. Selection of the customer for service is usually by some rule known as *queue discipline*. Service is then provided by the *service mechanism* after which the customer departs the queue.

A major fact of Markovian queueing models is that customers arrive in accordance with a Poisson process, thus all *interarrival times* are independent and exponentially distributed, and that all *service times* are also independent and exponentially distributed.

2.0.2 Parallel Mathematical Queue Definitions.

Consider a basic queueing system where arrivals to the system are a characterized by counting processes, $\alpha(t)$ and departures by $\delta(t)$. Both of these functions produce a

[6] B. S. Blanchard and W. J. Fabrycky, *Systems Engineering and Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990, pg. 271.

discrete change in the state of the system. Define the number of customers in the system at time t as: $S(t) = \alpha(t) - \delta(t)$, which corresponds to the state of the system. Let N(t) then denote the total number of customers that have arrived into the system. The input source can then be generated using one of many techniques depending on the probability distribution (or mass) function of the arrival process.

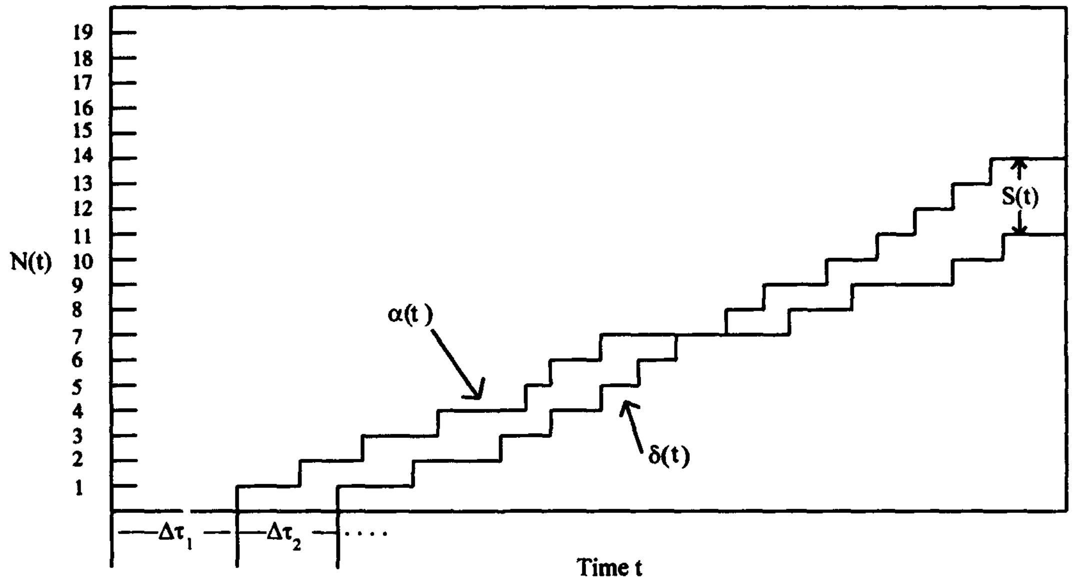


Figure 2.1 - Arrivals and Departures[7].

Section 1.3.1.2 described how a Poisson process can be modeled as a pooled process. For Markovian queues, the arrival and service process can also be generated as a pooled process. Let λ_i describe the arrival process and μ_i the service process of system i which is to be simulated on processor i. The incremental interevent time can then be generated using the well-known inversion technique. Suppose now that the parametric variations across all processors were generated by holding the arrival process constant and varying the service rates. Let μ_i be the service rate at the ith processor, i=1..N, and

[7] L. Kleinrock, *Queueing Systems Volume 1: Theory*, John Wiley & Sons, New York, 1975, pg. 16.

let $\mu_{max} = Max(\mu_i)$, i=1..N. Assuming that events will be marked on the front end before being broadcast to the individual processors, interevent times can be generated and used to update the system time on the front end by the following relationship:

$$\Delta\tau = -\frac{1}{\lambda + \mu_{max}} \ln(1 - U_1) \tag{2.1}$$

where U is a uniform [0.0,1.0] random variable. Applying the equations developed in Section 1.3.1.5, state changes are then *marked* on the front end as:

$$\Delta S(t) = \begin{cases} +1, \text{ if } U_2 \le \frac{\lambda}{\lambda + \mu_{max}} & \text{(arrival)} \\ -1, \quad \text{Otherwise} & \text{(departure)} \end{cases} \tag{2.2}$$

The actual state change is not applied at processor i until any required thinning is performed.

$$\Delta S_i(t) = \begin{cases} +1, & \text{if } \Delta S(t) = +1 \\ -1, & \text{if } \Delta S(t) = -1 \ \& \ U_3 \le \frac{\mu_i}{\mu_{max}} \\ 0, & \text{Otherwise} \end{cases} \tag{2.3}$$

Notice that three random numbers must be generated to perform front end marking. The final state updating step involves a check of the current system state. For a simple queue, there can be no fewer than 0 customers in the system at any time.

$$S_i(t_{now}) = S_i(t_{before}) + \begin{cases} \Delta S_i(t), & \text{if } S(t_{before}) > 0 \\ 0, & \text{Otherwise} \end{cases} \tag{2.4}$$

Now consider another queueing system in which the arrival process no longer requires thinning as the front end is now generating an unmarked process. Interevent times are generated using the same relationship given for the front end generation but are now stored at the local processors. Equations 2.2 and 2.3 are combined for both event marking and thinning on the back end.

$$\Delta S_i(t) = \begin{cases} +1, & \text{if } 0 \le U_2 < \dfrac{\lambda}{\lambda + \mu_{\max}} \\ -1, & \text{if } \dfrac{\lambda}{\lambda + \mu_{\max}} \le U_2 < \dfrac{\lambda + \mu_i}{\lambda + \mu_{\max}} \\ 0, & \text{Otherwise} \end{cases} \tag{2.5}$$

The last step once again is to determine if the attempted state transition is to an infeasible state. Equation 2.4 applies. Key is the fact that now only two random numbers are required to generate the next event type.

The event synchronous generation is performed in a slightly different fashion. Interevent times are now computed at processor i by the inversion technique. Once again, the arrival rate will be held constant across all processors. When $s_i(t) > 0$, then

$$\Delta\tau_i = -\frac{1}{\lambda + \mu_i} \ln(1 - U_1) \tag{2.6}$$

$$\Delta S_i(t) = \begin{cases} +1, & \text{if } U_2 \le \dfrac{\lambda}{\lambda + \mu_i} \\ -1, & \text{Otherwise} \end{cases} \tag{2.7}$$

If the system at processor i is in state 0, then Equations 2.6 and 2.7 become:

$$\Delta\tau_i = -\frac{1}{\lambda}\ln(1-U_1) \tag{2.8}$$

$$\Delta S_i(t) = +1 \tag{2.9}$$

The parametric interevent processes of the event synchronous simulations are similar in that a state transition occurs every time but the local system times of each are different. Essentially time is rescaled at each processor corresponding to the local event rates. Figure 2.2 provides a simple depiction of the associated time rescaling of the event synchronous method.

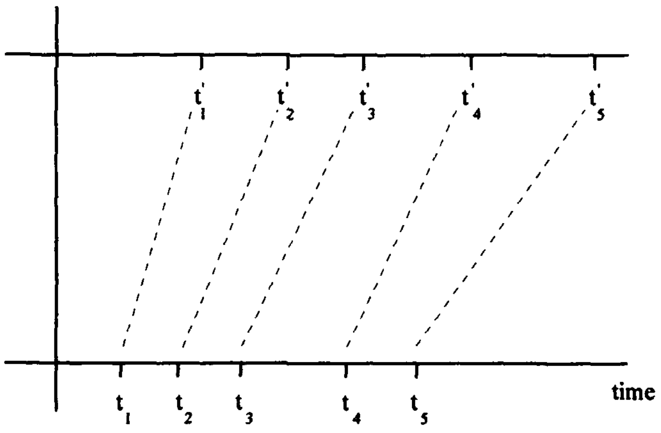


Figure 2.2 - Time Rescaling.

2.0.3 Queue Notation.

Most queues can be described by conventional labeling notation. The classification scheme is given as : $F_a/F_d/s[/k/p]$ where the symbols are as follows:

F_a - arrival (or interarrival) distribution.

F_d - departure (or service time) distribution.

s - number of parallel channels in the system.

k - maximum number (buffer size) allowed (in service plus waiting)

p - size of the population.

Figure 2.3 depicts a typical queueing system.

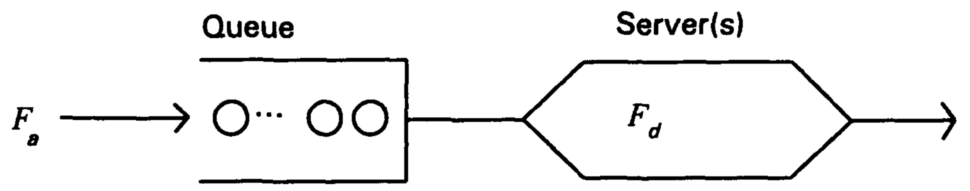


Figure 2.3 - General Queue.

The following are commonly used to replace the symbols F_a and F_d:

M - Poisson (exponential) arrival or departure distributions. M refers to the Markov property of the exponential distribution.

GI - general independent distribution of arrivals (or interarrival times).

G - general distribution of departures (or service times).

D - deterministic interarrival or service times.

E_k - Erlangian or gamma interarrival or service-time distribution with k phases.

The service discipline can also be described by one of the following:

FCFS - first come - first served.

LCFS - last come - first served.

SIRO - service in random order.

SPT - shortest processing (service) time.

GD - general service discipline.

M/M/1 then refers to a standard queueing system with exponential interarrival and service times with one server. The buffer size and population are both assumed to be infinite. The service discipline is always assumed to be *FCFS* unless otherwise specified.

There are many other queueing system variations that are not covered by this standard notation such as customer balking, state-dependent service times, and a wide range of other possible combinations. Also, series and networks of queues are not described in full detail with this notation. In general, the notation standardizes basic queueing principles into a readily understood format. Any deviations must be explained such that the system in question is fully understood.

2.0.4 Queue Performance Measures.

There are many performance measures that accurately describe different attributes of any queueing system. A basic measure is the *utilization ratio*, ρ. It is the ratio of the amount of work or customers entering the system to the maximum capacity of the system. Recall that s is the number of available servers, λ the arrival rate and μ the service rate. Then $\rho = \frac{\lambda}{s\mu}$; $s, \mu \neq 0$. If $\rho>1$, for a non-buffered system, the number of customers in the queue will tend to ∞ as the arrival rate exceeds service capacity.

Two important additional measures of interest are the *Mean Queue Length*, L, and the *Average Waiting Time*, W. The mean queue length describes the average number of customers in the queue and is defined as $L = E(S(t))$, the expected value of the system state, which includes customers in service. The average waiting time (including time in service) can be defined as $W = E(\frac{T}{N(T)})$, the expected value of the amount of time per customer in the system.

An important queueing system performance measure is the percentage of time the system may spend in any state. The percentage of time in state 0 which corresponds to the percentage of time the system is empty may be of interest. Also, another statistic of interest might be when S(t) > x, corresponding to the probability that there may be more than x customers in the system. Discrete Event Simulation exists for those systems in which analytical solutions may not exist.

2.1 Markovian Simulation Framework.

Uniformization of a Markov chain introduces a rate in which a state transitions to itself. The rate introduced at each state must be such that the transition rate at every state is the same. Vakili's standard clock method uses the front end marking technique to perform time synchronous simulations across a wide range of parametric variations. Many different processes are generated which have the same interevent times based on the maximum rate system, but different sample paths. In order for the reduced rate system to have the same system time as the maximum rate system, a Null rate is introduced which allows the system to transition from state i back to itself. Vakili's extension was to uniformize the rate not only across states within a system, but across multiple processors as well. The different parametric variations located on individual processors have synchronous system times but follow different sample paths based on local parameters.

Another alternative is then to generate an identical number of events across all of the processors and allow each to maintain its own individual system time as well as its current state information. A clear advantage to this method as opposed to the time synchronous method is that all events will now be used rather than some thinned away. All event synchronous parametric variations have different system times, but closely correlated sample paths.

The time synchronous (standard clock) and event synchronous parallel event generation methods provide both advantages and disadvantages in terms of simulation results. A key factor in determining which of the two methods to use is the goals of the simulation and what stopping criteria has been selected. Event synchronous allows a specified number of events to be generated across all parametric variations, whereas time synchronous provides the capability of selecting the system time at which to terminate all of the parametric variations.

2.1.1 Random Number Generation.

Critical to the success of any simulation is the generation of random variables. *Pseudo-Random Number Generators* have been used with great success. There are many classes of generators of which some are better than others. To improve computational speed, a generator should be used that has good statistical randomness qualities. It should also be capable of generating non-overlapping sequential random number values, which are also referred to as *random number streams*. Streams should be generated quickly and with as little computational effort as possible. The class of Linear Congruential Generators (LCGs) contain some of the most commonly used. They compute the *i*th integer X_i in the pseudo random sequence through the use of the recursive relationship: $X_i = (aX_{i-1} + c) \bmod m$. The values of a,c, and m determine the randomness qualities of the generator.

A full period LCG means that every integer between 0 and m-1 will occur only once before the sequence is repeated. There are three conditions that must hold.

(a) The only positive integer that exactly divides both m and c is 1.

(b) If q is a prime number that divides m, the q divides a-1.

(c) If 4 divides m, then 4 divides a-1.

Even if a LCG is full period, that fact does not necessarily preclude non-randomness. There are several statistical tests to determine the validity of any generator [BFS 1987]. The first is the test of *k-Dimensional Uniformity*. This test partitions a number sequence into non overlapping sequences of length k, and then places each sequence of numbers as a point within the k-dimensional cube, which is partitioned into s^k cells. A chi-square test is then performed on the number of points within each cell.

Another easily implemented test is called *Runs Up and Down.* This test counts the number of times the sequence changes direction. A hypothesis test is then performed stating that the random numbers are independent and identically distributed in accordance with the relevant mean and variance for the length of the sequence generated.

A third test, *Kolmogorov-Smirnov*, orders the sequence and computes a statistic based on the maximum positive and negative differences between the empirical and theorized cumulative distribution functions. A hypothesis test is then conducted to determine if the empirical data is drawn from the theorized cdf.

The LCG chosen for use is: $X_i = (16807X_{i-1} + 1)\text{mod } 2^{31} - 1$. This generator has full period as it meets the three conditions listed above and can be implemented in four lines of code on any machine like the CM-2 which works with 32 bits including the sign bit. This LCG only requires memory storage of the current values of X_i as each stream is generated. Random number starting seeds can be spaced far enough apart to ensure that the parallel streams do not overlap and are then independent.

The three previously mentioned statistical tests were run on this generator. All were all run on stream lengths of 1000, 10000 and 40000 generated values using successive multiple seeds. The LCG passed all three tests for each with a 95% confidence level. This generator is then *random* enough. LCGs are not the only generators that parallel simulations may use. Any *good* generator would provide similar randomness results.

2.1.2 The Alias Method.

The CM-2 identifies both processors and data elements as array members and uses array indices to identify both individual data elements as well as processing elements. The Alias Method [BFS 1987] is a means with which a uniform [0,1] random variable may be transformed into a discrete random variable. This can be turned into a major advantage when using the SIMD architecture. There is a memory storage requirement of two arrays of values. A uniform random variable is generated on the interval (0,n) by V=nU, where n is the number of discrete events that can occur. V is then truncated and its truncated portion, W, (now uniform (0,1)) is compared to the first array data element indexed by V. If W≤ than data element's particular value, then that event will be scheduled. Otherwise, the event indexed by the second array member (its alias) will be scheduled.

The two arrays are quickly set up during the initialization portion of a simulation. The major advantage is that only two array lookups have to occur in order to determine the next event type. If this method were not used, then on average, log(x) comparisons of U with the possible individual event probabilities would have to be made to determine the next event type, given there are x possible event types/ state transitions. Equation 2.5 depicts a small example of the number of comparisons that must be made.

The CM-2 has limited memory storage for each individual processor. Consider a general Markov chain in which every $P_{ij} \neq 0$. Then two n by n matrices (one real and one integer) have to be stored in each local processor's memory. For large Markov Chains and a large virtual processor ratio, the memory requirement could quickly exceed resources. Sparse matrix techniques could be employed if a majority of the $P_{ij}=0$ for a large system.

2.2 Event Synchronous Simulations.

The event synchronous event generation method is ideal for use when the performance of all parametric variations is to be compared after a fixed number of events. Consider a M/M/1/k queue. State transitions can occur from states 0 to 1,.., (k-1) to k, as well as k to (k-1),.., 1 to 0. For this system, there are two possible events, an arrival or a departure from the system. A pooled Poisson process (Section 1.3.1.2) can be used to generate an event rate of both types, $(\lambda+\mu)$. On a CM-2, that means that two Uniform (0,1)'s can be used to generate all parametric variations. Interevent times are computed using the first Uniform (0,1) and the following equation using inversion.

$$\Delta\tau_i = -\frac{1}{\lambda_i + \mu_i}\ln(1-U_1) \tag{2.10}$$

System times are stored on the local processors. Once the interevent time is determined, the pooled process must be marked into the actual event type at each individual processor using the following probability formulation. No thinning occurs at any processor.

$$P(\text{Arrival})\big|_i = \frac{\lambda_i}{\lambda_i + \mu_i} \tag{2.11}$$

$$P(\text{Departure})\big|_i = \frac{\mu_i}{\lambda_i + \mu_i} \tag{2.12}$$

Recall that for this method, if the system state is in either 0 or k, then the next appropriate event will be generated automatically. Otherwise, events are accepted in accordance with Equations 2.11 and 2.12 and the local system state updated. A state diagram for an example M/M/1/3 queue illustrates the relevant rates with which the system makes state transitions.

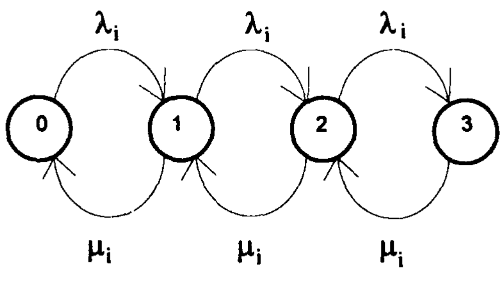


Figure 2.4 - Event Synchronous State Diagram M/M/1/3.

The following equations depict the relative probability that a particular state transition will occur given that the system is in that particular state upon generation of an interevent for the general M/M/1/k queue.

$$P_{0,1} = 1 \tag{2.13}$$

$$P_{k,k-1} = 1 \tag{2.14}$$

$$P_{j,j+1}\Big|_{\text{Processor } i} = \frac{\lambda_i}{\lambda_i + \mu_i}, \text{ j=1..k.} \tag{2.15}$$

$$P_{j+1,j}\Big|_{\text{Processor } i} = \frac{\mu_i}{\lambda_i + \mu_i}, \text{ j=0..k-1.} \tag{2.16}$$

Note once again that transitions from state 0 to 1 and from state k to k-1 will occur with probability 1. Two binary variables are now introduced to adjust the interevent time computation and update the system time when the system is in a *boundary* condition.

$I_0 = 0$ if the system is in state 0, 1 otherwise. Likewise, $I_k = 0$ if the system is in state k, 0 otherwise. Equation 2.10 now becomes:

$$\Delta t_i = -\frac{1}{\lambda_i I_{k,i} + \mu_i I_{0,i}} \ln(1-U) \tag{2.17}$$

The following algorithm provides the CM-2 implementation of the Event Synchronous method. Steps in **bold** signify that the instruction must be done across all parallel processors simultaneously or on the back end. The algorithm does not differ if more than one replication is performed simultaneously. System time still must be stored at every parallel processor. Event probabilities are initially computed and broadcast to the processors using Equations 2.11 and 2.12.

INITIALIZE:	Initialize Event Counter, **Set State=0.**
	Initialize Arrival Counter, Time Clock.
	Broadcast Parametric Variation Data Sets to Processors.
	Compute Event Probabilities.
	Initialize Alias Arrays.
	Open Output Files.
LOOP:	**Generate 2 Uniform Random Variables [0,1]**
	Compute Interevent Time using U_2.
	Let $V = \lfloor nU_1 \rfloor$
	Let $W = nU_1 - V$
	If State=0; Perform Arrival.
	Else If State=k; Perform Departure.
	Else (Compare W with Alias Array value indexed by V,
	If $U_1 \leq W$, Perform that Event
	Else, Perform Aliased Event).
	ENDIF
	Update Raw Performance Statistics.
	Update State.
	Increment Counter.
	Check Stopping Criteria,
	If Met, Go To WRAP-UP,
	Else, Go To LOOP.
WRAP-UP:	**Compute Performance Statistics.**
	Write Output Files.

For this simple queueing system, it is very easy to maintain running totals of raw performance statistics. Both the Mean Queue Length and the Mean Waiting Time can be incrementally computed during every pass of LOOP of the simulation. Essentially, the raw statistics are: $L_{raw,i} = W_{raw,i} = \sum S_i \Delta t_i$, where S_i is the local system state. Then, $L_i = \frac{\sum S_i \Delta t_i}{T_i}$ and $W_i = \frac{\sum S_i \Delta t_i}{N_i}$. It is therefore necessary to store both the local system time and the number of arrivals to the system at every processor in order to compute these statistics.

2.3 Time Synchronous Simulations.

The time synchronous implementation of the M/M/1/k queue can be accomplished in any of four ways to ensure that the system time is synchronous across all parametric variants. Either holding the arrival rate, λ, constant or the service rate, μ, constant may be generated by either a marked or an unmarked Poisson process from the front end. There are also two hybrid methods in which both λ and μ are varied that differ by where event marking takes place.

Setting up the first time synchronous method entails holding the arrival rate constant and varying the service rates, μ_i, to form the parametric variations. Let $\mu_{max} = \text{Max}(\mu_i), i = 1..N$. If events are generated on the front end, the appropriate probability formulation is:

$$P(\text{Arrival}) = \frac{\lambda}{\lambda + \mu_{max}} \tag{2.18}$$

$$P(\text{Departure}) = \frac{\mu_{max}}{\lambda + \mu_{max}} \tag{2.19}$$

The only events requiring thinning at the local processors are departure events. The local thinning probability is then:

$$P(\text{Accept Departure})\Big|_i = \frac{\mu_i}{\mu_{max}} \tag{2.20}$$

Generating events on the back end entails broadcasting the appropriate event probabilities to the local processors. The probability formulation is the same but now, Null events have their own explicit probability of being generated.

$$P(\text{Arrival})\Big|_i = \frac{\lambda}{\lambda + \mu_{max}} \tag{2.21}$$

$$P(\text{Departure})\Big|_i = \frac{\mu_i}{\lambda + \mu_{max}} \tag{2.22}$$

$$P(\text{Null})\Big|_i = \frac{\mu_{max} - \mu_i}{\lambda + \mu_{max}} \tag{2.23}$$

The state diagram at processor i for the M/M/1/3 is given for either of the time synchronous methods of holding λ constant. Notice that the departure from each state has a uniformized rate. Also note that the system will attempt transitions to states '-1' and 'k+1' which do not exist so that the system will actually stay in state 0 or k. The rates depicted are for both the front end and back end marking methods.

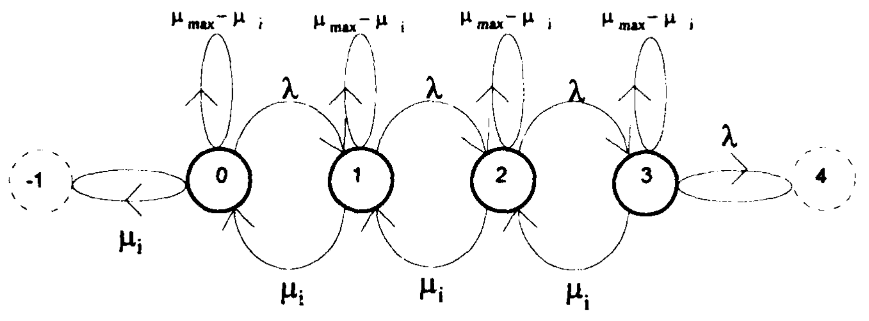


Figure 2.5 - Time Synchronous State Diagram M/M/1/3 - Arrival Rate Constant.

The following equations reflect the actual state transition probabilities for the M/M/1/k system at processor i. λ and μ_i are the local system parameters.

$$P_{0,0} = \frac{\mu_{max}}{\lambda + \mu_{max}} \tag{2.24}$$

$$P_{k-1,k}\Big|_{Processor\ i} = \frac{\lambda + \mu_{max} - \mu_i}{\lambda + \mu_{max}} \tag{2.25}$$

$$P_{j,j}\Big|_{Processor\ i} = \frac{\mu_{max} - \mu_i}{\lambda + \mu_{max}}, \text{j=1..k-1.} \tag{2.26}$$

$$P_{j,j+1} = \frac{\lambda}{\lambda + \mu_{max}}, \text{j=0,k-1.} \tag{2.27}$$

$$P_{j+1,j}\Big|_{Processor\ i} = \frac{\mu_i}{\lambda + \mu_{max}}, \text{j=1..k.} \tag{2.28}$$

The next set of time synchronous event generation methods entail holding the service rate constant and varying the arrival rate across the processors. Let

$\lambda_{max} = \mathrm{Max}(\lambda_i), i = 1..N$. If events are marked on the front end, the probability formulation is then:

$$P(\mathrm{Arrival}) = \frac{\lambda_{max}}{\lambda_{max} + \mu} \tag{2.29}$$

$$P(\mathrm{Departure}) = \frac{\mu}{\lambda_{max} + \mu} \tag{2.30}$$

Events requiring thinning at the local processors are arrival events. The local thinning probability is then:

$$P(\mathrm{Accept\ Arrival})|_i = \frac{\lambda_i}{\lambda_{max}} \tag{2.31}$$

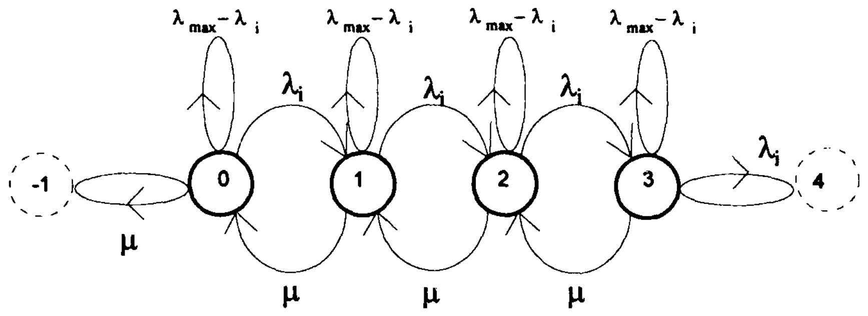
Similar to the previous back end marking method, the appropriate event probabilities must be broadcast to the local processors for their use. The probability formulation is the same but once again, Null events have their own explicit probability of being generated.

$$P(\mathrm{Arrival})|_i \frac{\lambda_i}{\lambda_{max} + \mu} \tag{2.32}$$

$$P(\mathrm{Departure})|_i = \frac{\mu}{\lambda_{max} + \mu} \tag{2.33}$$

$$P(\mathrm{Null})|_i = \frac{\lambda_{max} - \lambda_i}{\lambda_{max} + \mu} \tag{2.34}$$

The state diagram at processor i for the M/M/1/3 is given for either of the time synchronous methods of holding μ constant. The rates depicted are for both the front end and back end marking methods. This method is essentially the dual of holding the arrival rate constant.



$\lambda_{max} - \lambda_i$
$\lambda_{max} - \lambda_i$
$\lambda_{max} - \lambda_i$
$\lambda_{max} - \lambda_i$
λ_i
λ_i
λ_i
λ_i
-1
0
1
2
3
4
μ
μ
μ
μ

Figure 2.6 - Time Synchronous State Diagram M/M/1/3 - Service Rate Constant.

The following equations reflect the actual state transition probabilities for the M/M/1/k system at processor i. λ_i and μ are now the local system parameters.

$$P_{0,0}\Big|_{\text{Processor } i} = \frac{\lambda_{max} + \mu - \lambda_i}{\lambda_{max} + \mu} \tag{2.35}$$

$$P_{k-1,k} = \frac{\lambda_{max}}{\lambda_{max} + \mu} \tag{2.36}$$

$$P_{j,j}\Big|_{\text{Processor } i} = \frac{\lambda_{max} - \lambda_i}{\lambda_{max} + \mu}, \text{ j=1..k-1.} \tag{2.37}$$

$$P_{j,j+1}\Big|_{\text{Processor } i} = \frac{\lambda_i}{\lambda_{max} + \mu}, \text{ j=0..k-1.} \tag{2.38}$$

$$P_{j+1,j} = \frac{\mu}{\lambda_{\max} + \mu}, \text{ j=1..k.} \tag{2.39}$$

Consider now a parametric set of systems in which the interevent rate across all processors is the same. Both the arrival and service rates can be varied such that their sum is the same. Define r as the system interevent rate and set it to some constant value. Now solve for the arrival and service rates by using the relationship: $r = \lambda_i + \mu_i$. Front end marking requires an actual event type generated to be broadcast to the individual processors. Therefore, interevents must be generated at an appropriate rate. If R is the actual rate that events are generated on the front end, then it is determined using the following relationship:

$$R = \text{MAX}(\lambda_i) + \text{MAX}(\mu_i) \forall i \tag{2.40}$$

This *hybrid* method then generates overall events at a faster rate than the other time synchronous methods. Once again, events will be thinned. It is obvious that the rate at which events will be thinned and hence not used is R-r.

Events are marked on the front end with the following probability.

$$P(\text{Arrival}) = \frac{\lambda_{\max}}{\lambda_{\max} + \mu_{\max}}. \tag{2.41}$$

$$P(\text{Departure}) = \frac{\mu_{\max}}{\lambda_{\max} + \mu_{\max}} \tag{2.42}$$

Back end thinning at processor i now occurs for both of the front end marked events.

$$P(\text{Accepting Arrival}|\text{Arrival is Broadcast})\Big|_i = \frac{\lambda_i}{\lambda_{max}} \tag{2.43}$$

$$P(\text{Accepting Departure}|\text{Departure is Broadcast})\Big|_i = \frac{\mu_i}{\mu_{max}} \tag{2.44}$$

The state diagram at processor i for the M/M/1/3 is given for the front end hybrid generation scheme. The null event rate is given by: $R - r = \lambda_{max} + \mu_{max} - \lambda_i - \mu_i$

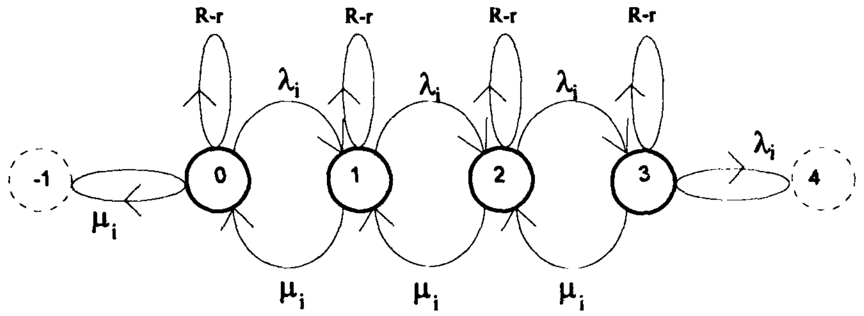


Figure 2.7 - Time Synchronous State Diagram M/M/1/3 - Front End Hybrid.

The following equations reflect the actual state transition probabilities for the M/M/1/k system at processor i. λ_i and μ are the local system parameters.

$$P_{0,0}\Big|_{\text{Processor } i} = \frac{\lambda_{max} + \mu_{max} - \lambda_i}{\lambda_{max} + \mu_{max}} \tag{2.45}$$

$$P_{k,k}\Big|_{\text{Processor } i} = \frac{\lambda_{max} + \mu_{max} - \mu_i}{\lambda_{max} + \mu_{max}} \tag{2.46}$$

$$P_{j,j}\Big|_{\text{Processor } i} = \frac{\lambda_{max} + \mu_{max} - \lambda_i - \mu_i}{\lambda_{max} + \mu_{max}}, \text{j=1..k-1} \tag{2.47}$$

$$P_{j,j+1}\Big|_{\text{Processor } i} = \frac{\lambda_i}{\lambda_{max} + \mu_{max}}, j=1..k-1. \tag{2.48}$$

$$P_{j,j+1}\Big|_{\text{Processor } i} = \frac{\mu_i}{\lambda_{max} + \mu_{max}}, j=0..k-1. \tag{2.49}$$

An alternative hybrid method involves varying both the arrival and service rates but broadcasting unmarked events from the front end. The principle advantage of this method is that now the maximum interevent rate may be generated at every processor, thus removing the possibility of any null events. Time is still synchronous as the event generation rate is the same.

$$P(\text{Arrival})\Big|_i = \frac{\lambda_i}{\lambda_i + \mu_i} \tag{2.50}$$

$$P(\text{Departure})\Big|_i = \frac{\mu_i}{\lambda_i + \mu_i} \tag{2.51}$$

The state diagram at processor i for the M/M/1/3 is given for the front end hybrid generation scheme. Note the similarity to the event synchronous method with the major difference of attempted events to infeasible states.

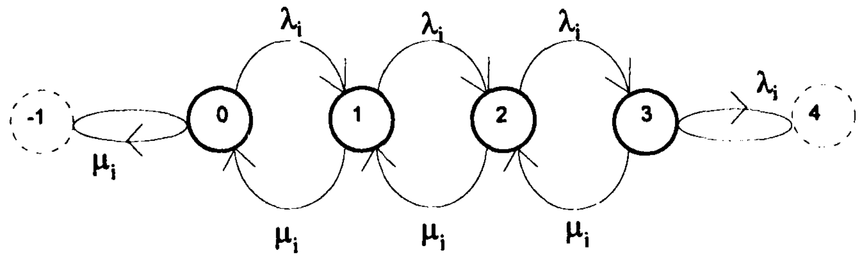


Figure 2.8 - Time Synchronous State Diagram M/M/1/3 - Back End Hybrid.

The state transition probabilities also reflect the non-existence of any Null events. Only transitions from state 0 to 0 and k to k reflect the difference between the this method and the event synchronous method.

$$P_{0,0}\Big|_{\text{Processor i}} = \frac{\mu_i}{\lambda_i + \mu_i} \tag{2.52}$$

$$P_{k,k}\Big|_{\text{Processor i}} = \frac{\lambda_i}{\lambda_i + \mu_i} \tag{2.53}$$

$$P_{j,j+1}\Big|_{\text{Processor i}} = \frac{\lambda_i}{\lambda_i + \mu_i}, \text{ j=1..k-1.} \tag{2.54}$$

$$P_{j,j+1}\Big|_{\text{Processor i}} = \frac{\mu_i}{\lambda_i + \mu_i}, \text{ j=0..k-1.} \tag{2.55}$$

All of the time synchronous methods with the exception of the back end hybrid compute the same interevent time, $\Delta\tau$, based on the maximum event rate of all the parametric variations. If the arrival rate is being held constant then $\lambda_{max} = \lambda$.

$$\Delta\tau = -\frac{1}{\lambda_{\text{max}} + \mu_{\text{max}}}\ln(1-U) \tag{2.56}$$

The back end hybrid time synchronous method still computes the same interevent time, $\Delta\tau$, however it is now not based on the maximum event rate of all the parametric variations but on the constant rate selected for the simulations.

$$\Delta\tau = -\frac{1}{\lambda_i + \mu_i}\ln(1-U) = -\frac{1}{r}\ln(1-U) \tag{2.57}$$

The next algorithm is the CM-2 implementation of the Standard Clock/ Front End Poisson process event generation.

INITIALIZE:	Initialize Event Counter, **Set State=0.**
	Initialize Arrival Counter, Time Clock.
	Broadcast Parametric Variation Data Sets to Processors.
	Compute Front End Marking Probabilities using Appropriate Equations.
	Compute Probabilities of Accepting Events, P_e.
	Initialize Alias Arrays.
	Open Output Files.
LOOP:	Generate 3 Uniform [0,1]
	Compute Interevent Time using U_3
	* Note that this step can be done on the front end if only one replication is being performed.
	Let $V = \lfloor nU_1 \rfloor$
	Let $W=nU_1-V$
	Compare W with Alias Array value indexed by V,
	If $U_1 \leq W$, Perform that Event
	Else, Perform Aliased Event).
	Compare U_2 with Local Event Probability, P_e
	If $U_2 \leq P_e$, Perform Event
	Else, No State Change
	Now Check to see if State 0 or State k has been violated,
	If so, Reset System State to Previous State
	[illegible]date Raw Performance Statistics.
	[illegible]pdate State.
	[illegible]ement Counter.
	Check Stopping Criteria,
	If Met, Go To WRAP-UP,
	Else, Go To LOOP.
WRAP-UP:	**Compute Performance Statistics.**
	Write Output Files.

The primary difference between the time synchronous and event synchronous algorithms, besides the time synchronicity is the fact that now, the system state is checked to see if a boundary state has been violated. If so, then the state is reset to its previous state. Raw performance computation is the same as in the event synchronous method.

The following algorithm provides the CM-2 implementation of any of the back end time synchronous generation methods. Once again, steps in **bold** signify that it will be done across all processors simultaneously. This algorithm is adjusted so that more than one replication of each parametric variant will be done simultaneously and that system times are stored at local processors.

INITIALIZE:	Initialize Event Counter, **Set State=0.**
	Initialize Arrival Counter, Time Clock.
	Broadcast Parametric Variation Data Sets to Processors.
	Compute Event Probabilities using Appropriate Equations.
	Initialize Alias Arrays.
	Open Output Files.
LOOP:	**Generate 2 Uniform [0,1]**
	Compute Interevent Time * using U_2
	Let $V = \lfloor nU_1 \rfloor$
	Let $W=nU_1-V$
	Compare W with Alias Array value indexed by V,
	If $U_1 \leq W$, Perform that Event
	Else, Perform Aliased Event).
	Now Check to see if State 0 or State k has been violated,
	If so, Reset System State to Previous State
	Update Raw Performance Statistics.
	Update State.
	Increment Counter.
	Check Stopping Criteria,
	If Met, Go To WRAP-UP,
	Else, Go To LOOP.
WRAP-UP:	**Compute Performance Statistics.**
	Write Output Files.
	* Note that this step can be executed on the front end if only one replication is being performed.

Model equivalency can be confirmed by computing the limiting probabilities of being in any state. $\pi_{i,m} = \sum_{j=0}^{k} \pi_{i,j} P_{jm}$ and $\sum_{m=0}^{k} \pi_{i,m} = 1$ provide the necessary relationships, where i identifies the individual processor, and *jm* denotes a transition from state j to m. Solving all of the time synchronous models independently, produces the same limiting probabilities. The event synchronous method produces a different set of limiting probabilities. The π_i's represent the proportion of transitions that take the system into state i. However, for the event method, the mean time spent in each state is not the same as compared to the uniformized time synchronous methods. When a local processor is in either state 0 or k, the interevent time is adjusted to compensate for the next deterministic event type. The following relationship weights proportionally each limiting probability by the mean time spent in each state, thus converting the discrete time Markov chain to continuous time.

$$P_i = \frac{\frac{\pi_i}{\nu_i}}{\sum_j \frac{\pi_j}{\nu_j}}; \quad \nu_0 = \frac{1}{\lambda_i}; \; \nu_k = \frac{1}{\mu_i}; \; \nu_i = \frac{1}{\lambda_i + \mu_i}, \; i = 1..k-1$$ for the event synchronous method. Incorporating these mean times, the limiting probabilities P_i = the same limiting probabilities for the time synchronous methods ensuring that the same systems are being simulated.

3.0 Efficiency.

One of the disadvantages of the time synchronous method is that any number of generated events are not used by the parallel processors during each pass of the main simulation event generation loop. Chapter 3.0 defines both the individual system as well as the average overall simulation efficiency associated with the various time synchronous implementations of both the M/M/1 and M/M/1/k queueing systems. The concepts developed in this chapter provide the basis for variance prediction in Chapter 4.0

An important characteristic of a simulation's performance is the resulting variance of any performance measure. Maximum utilization of all available processors during a parallel simulation should ultimately lead to the lowest possible variance of the performance measures and the highest possible speedup associated with the simulations. Certain parallel simulation techniques such as the time synchronous/ standard clock algorithms tend to allow processors to sit idle while others are processing. Overall speedup is then a function of the event usage or efficiency of the processors. Consider two systems simulated in parallel in which 200 and 190 events respectively are generated using the time synchronous/ standard clock method. Assume that the stopping criteria for these two simulations is when 200 events are generated on the front end. Therefore, the second system had 10 events *thinned* away. Processor efficiency of the first processor is then 1.0 or 100%. Processor efficiency of the second processor then is $\frac{200+190}{200+200} = 0.975$ or 97.5%. An additional simulation time increment would then be required to bring system number 2 up to a total of 200 events. Overall processor efficiency, of both processors 1 and 2, would just be the average of the two or 98.75%. This section analyzes the overall processor efficiency of the standard clock algorithm and provides an analytical formulation for the associated speedup penalty. It also develops a basis for variance estimation of system performance measures of the time synchronous method as compared to the event synchronous generation scheme.

3.1 Simulation Efficiency - Null Events.

The standard clock algorithm generates a marked Poisson process at the front end and *thins* or keeps events at each local processor with probabilities commensurate with the local event rates. Events that are thinned are essentially null events as the system clock is updated but no state transitions are made. Figure 3.1 depicts the thinning phenomena at processor i.

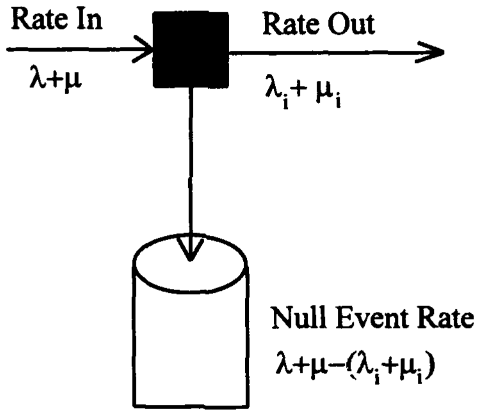


Figure 3.1 - Null Event Splitting/ Thinning.

If the output rate $(\lambda_i+\mu_i)$ is very low in comparison to the input, then most of the events will be thinned away. From a parallel simulation viewpoint, this could be computationally very inefficient when viewed across all the processors. While some individual processors may be generating actual events, others will be idle.

If we assume that during a clock cycle of a simulation run, the system time and state are updated, then for those events in which the state is not updated, that processor's capability is *wasted.* For any of the time synchronous schemes, the system time is stored on the front end. Efficiency, ε, is then defined as:

$$\varepsilon = \frac{\text{Number of active processors during a clock cycle}}{\text{Total number of available processors}}$$

Poorly choosing the thinning parameters could lead to much wasted computational effort on the part of the majority of the individual processors.

For a M/M/1 queue, two types of events can occur, a customer arrival or a customer departure. From Section 1.3.1.3, it was shown that the combined maximum rate of both event types could be marked into individual event types on the front end and subsequently thinned to the system of interest at the individual processors. Recalling the definition of the utilization ratio of a queueing system, the following additional relationships are now introduced:

$$\rho_{\min} = \min\left(\frac{\lambda_i}{\mu_i}\right) \equiv \frac{\lambda_{\min}}{\mu_{\max}} \tag{3.1}$$

$$\rho_{\max} = \max\left(\frac{\lambda_i}{\mu_i}\right) \equiv \frac{\lambda_{\max}}{\mu_{\min}} \tag{3.2}$$

Assume that $\rho_{\min} > 0$, and $\rho_{\min} \le \rho_i \le \rho_{\max}$. Further assume $\rho_{\max} < 1$ for the M/M/1 queue, else the system will tend to fill up as arrivals will exceed departures.

Let $\rho_{\max}$ and $\rho_{\min}$ define the upper and lower limits of the range of the M/M/1 queues to be simulated. The N local processor variants could be then spaced evenly $\frac{\rho_{\max} - \rho_{\min}}{N-1}$ apart. Rather than uniformly spread the range, there are many other schemes under which the local parameter values can be initialized but are not discussed in this chapter.

Define R as the front end maximum rate of event generation. Either λ or μ could be held constant or both could be varied across all processors with events marked on the front end.

$$R = \lambda_{max} + \mu_{max} \tag{3.3}$$

Equations 3.1 and 3.2 are used to solve for the range of service completion or arrival rates respectively. Define r as the effective local processor event generation rate. Table 3.1 depicts an example of how the arrival rates and service rates are computed by holding either λ or μ constant using an event generation rate, $r = 1$ at every local processor. There are also two hybrid cases in which both the arrival and service rates are varied. Note how the front end event generation rate of the front end hybrid, is greater than that of the other parameterization schemes.

	R	ρ_{min}	ρ_{max}	λ_{min}	λ_{max}	μ_{min}	μ_{max}
λ Constant	1.0	0.1	0.9	0.0909	0.0909	0.1010	0.9091
μ Constant	1.0	0.1	0.9	0.0526	0.4737	0.5263	0.5263
Back End Hybrid	1.0	0.1	0.9	0.0909	0.4737	0.5263	0.9091
Front End Hybrid	1.3828	0.1	0.9	0.0909	0.4737	0.5263	0.9091

Table 3.1 - Rate Computation.

The first three cases produce overall events at the same rate and simulate the same queues by virtue of the range of the simulated utilization ratios which have been varied over the range of interest. λ or μ constant can be simulated by the front end computer producing either marked or non-marked Poisson events. The front end hybrid case contains the event rates that would be used in the implementation of the Standard Clock where the marked Poisson process is produced on the front end. Its effective event local processor generation rate, r = 1.0.

Recall that a Null event is any event that is thinned or does not produce a state transition. We are then interested in the ratio of the number of null events to the total number of events generated by redefining $\varepsilon = 1 - \frac{\text{Number of Null Events}}{\text{Total Number of Events}} = 1 - P(\text{Null})$. The first step is determining how many Null events can be expected.

3.1.1 Null Events - Arrival Rate Constant.

From Section 2.3, the probability of a Null event for holding λ constant was developed.

$$P(\text{Null},\lambda)\big|_i = \frac{\mu_{max} - \mu_i}{\lambda + \mu_{max}} \tag{3.4}$$

Rearranging Equation 3.1; $\mu_{max} = \frac{\lambda}{\rho_{min}}$. Similarly, $\mu_i = \frac{\lambda}{\rho_i}$. Solving Equation 3.4 in terms of ρ by factoring out the constant λ yields:

$$P(\text{Null},\lambda)\big|_i = \frac{\rho_i - \rho_{min}}{\rho_i(1+\rho_{min})} \tag{3.5}$$

3.1.2 Null Events - Service Rate Constant.

Now consider the same range of systems but hold the service rate constant and vary the arrival rates. The probability of a Null event is similar to that developed in Section 3.1.1.

$$P(\text{Null},\mu)\big|_i = \frac{\lambda_{max} - \lambda_i}{\lambda_{max} + \mu} \tag{3.6}$$

Rearranging this equation by applying $\lambda_{max} = \rho_{max}\mu$ and $\lambda_i = \rho_i\mu$ and factoring out μ yields:

$$P(\text{Null},\mu)\big|_i = \frac{\rho_{max} - \rho_i}{\rho_{max} + 1} \tag{3.7}$$

3.1.3 Null Events - Back End Hybrid.

For this case, recall that a set of arrival and service rates across all processors can be formulated such that their sum is a constant. Unmarked events are then generated on the front end at the rate, R, and sent to the back end for event marking. Essentially, $R = \lambda_i + \mu_i \forall i$. The major difference now is that events are generated at each parallel processor at the same rate and no thinning is required.

$$P(\text{Null},\text{BE Hybrid})\big|_i = 0 \tag{3.8}$$

3.1.4 Null Events - Front End Hybrid.

This case has a different overall event generation rate at the front end and across all of the processors. Generating marked events on the front end and varying both the local arrival and service rates such that their sum is a constant produces a faster interevent generation rate at the front end than the individual processors. Effectively, thinning occurs as a result of the difference in these rates. Marked events are generated on the front end at rate $\lambda_{max} + \mu_{max}$ and accepted at rate $\lambda_i + \mu_i$ at the local processors. Events then are not accepted at the back end processors with the following probability:

$$P(\text{Not Accepted},\text{FE Hybrid})\big|_i = \frac{\lambda_{max} + \mu_{max} - \lambda_i - \mu_i}{\lambda_{max} + \mu_{max}} \tag{3.9}$$

Not accepted events can also be considered the same as Null events generated during back end marking. To rearrange the equation in terms of ρ, requires the use of several relationships. Recall that the effective back end rate $r = \lambda_i + \mu_i \forall i$.

Substituting $\mu_i = \frac{\lambda}{\rho_i}$ yields: $\lambda_i + \frac{\lambda_i}{\rho_i} = r$. Therefore $\lambda_i = \frac{r\rho_i}{1+\rho_i}$ and $\mu_i = \frac{r}{1+\rho_i}$.

Solving for the relationships of λ_{max} and μ_{max} in terms of ρ is a bit more complicated. The extreme limits of the back end hybrid are the same as either λ or μ constant and can be observed in Table 3.1. Therefore μ_{max} can be solved by holding λ constant and vice versa. By using the front end generation rate, R, holding λ constant and solving for μ_{max} yields:

$\lambda + \mu_{max} = R$, $\lambda = \rho_{min}\mu_{max}$; then $\rho_{min}\mu_{max} + \mu_{max} = R$; therefore, $\mu_{max} = \frac{R}{1+\rho_{min}}$.

Similarly, by holding μ constant, λ_{max} can be solved for.

$\lambda_{max} + \mu = R$, $\mu = \frac{\lambda_{max}}{\rho_{max}}$; then $\lambda_{max} + \frac{\lambda_{max}}{\rho_{max}} = R$, therefore, $\lambda_{max} = \frac{R\rho_{max}}{1+\rho_{max}}$.

The terms of Equation 3.9 are then substituted with the following relationships:

Letting, $\lambda_{max} = \frac{\rho_{max}}{1-\rho_{max}}$, $\mu_{max} = \frac{1}{1-\rho_{min}}$, $\lambda_i = \frac{\rho_i}{1-\rho_i}$, and $\mu_i = \frac{1}{1-\rho_i}$.

Rearranging and solving Equation 3.9 in terms of ρ yields:

$$P(\text{Not Accepted Event}, \text{FE Hybrid})\big|_i = \frac{\rho_{max} - \rho_{min}}{1 + 2\rho_{max} + \rho_{max}\rho_{min}} \tag{3.10}$$

Notice that the P(Not Accepted Event) is constant throughout the chosen range of ρ's.

Figure 3.2 provides a graphical depiction of the local processor efficiency for two ranges of M/M/1 queues where ρ=[0.1,0.9] and [0.01,0.99] for each of the different time synchronous parameterization methods.

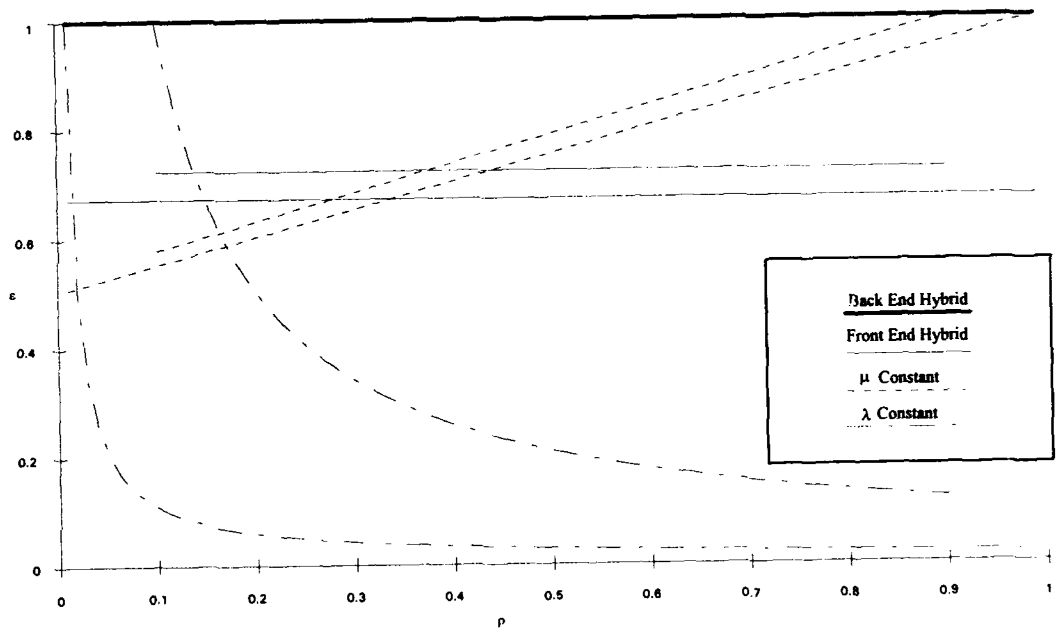


Figure 3.2 Individual Processor Efficiency.

Processor efficiency starts at 100% for λ constant and rapidly drops off. Note the substantial difference between ρ_{min}= 0.01 and 0.1 for this case. Efficiency for μ constant is linear and culminates at 100%. For the back end hybrid case, efficiency is constant at 100% as no events are thinned away. Efficiency is also constant for the front end hybrid case but is now a function of the limits of the range of ρ.

Averaging the processor efficiency over the full range of simulated systems provides a measure as to the overall processor utilization. Define $\bar{\varepsilon}$ as the overall average processor efficiency and N be the number of parallel processors.

$$\bar{\varepsilon} = 1 - \frac{1}{N}\sum_{i=1}^{N} \left.\frac{\text{Number of Null Events}}{\text{Total Number of Events}}\right|_i = 1 - \frac{1}{N} \text{P(Null)}\big|_i \tag{3.11}$$

If an infinite number of processors were available, then Equations 3.5, 3.7, and 3.10 can be integrated to solve for $\bar{\varepsilon}$.

3.1.5 Average Processor Efficiency - *Arrival Rate Constant.*

A major simplifying assumption is that the range to be simulated is spread uniformly across all the processors. Multiplying by the constant $\frac{1}{(\rho_{max} - \rho_{min})}$ is equivalent to multiplying by $\frac{1}{N}$ in Equation 3.11. If λ is held constant then:

$$\bar{\varepsilon}(\text{Null}, \lambda) = 1 - \frac{1}{(\rho_{max} - \rho_{min})} \int_{\rho_{min}}^{\rho_{max}} \frac{\rho - \rho_{min}}{\rho(1 + \rho_{min})} d\rho \tag{3.12}$$

Integrating Equation 3.12 yields:

$$\bar{\varepsilon}(\text{Null}, \lambda) = 1 - \frac{(\rho_{max} - \rho_{min} - \rho_{min} \ln(\frac{\rho_{max}}{\rho_{min}}))}{(\rho_{max} - \rho_{min})(1 + \rho_{min})} \tag{3.13}$$

Figure 3.3 provides a graphical representation of Equation 3.13. ρ_{min} and ρ_{max} were incremented in steps of 0.05 each within the range [0.05,0.90]. The highlighted point represents a utilization range of [0.1,0.9] and depicted in Figure 3.2 with an average efficiency of 0.341. To use this figure, enter the x-axis with the ρ_{min} of interest. Move up until the appropriate ρ_{max} curve is intersected. Read average efficiency from the y-axis.

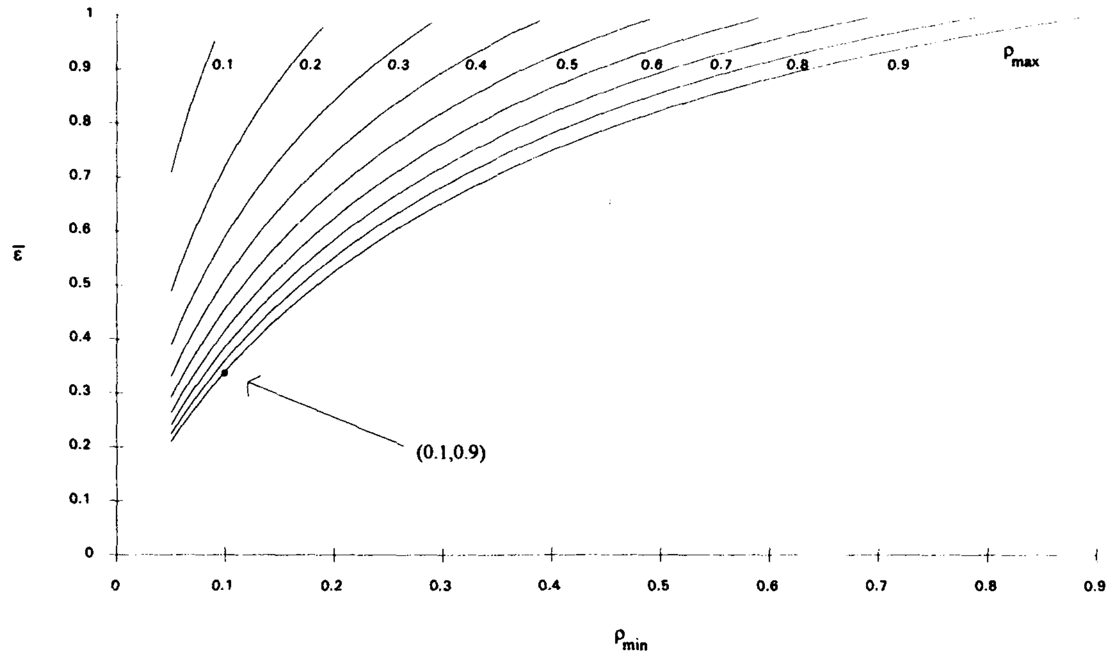


Figure 3.3 - Average Efficiency, Arrival Rate Constant.

3.1.6 Processor Efficiency - Service Rate Constant.

Average efficiency computation for the case in which the service rates are held constant is computed in the same manner as the previous case. Integrating Equation 3.7 yields:

$$\bar{\varepsilon}(\text{Null},\mu) = 1 - \frac{1}{(\rho_{max} - \rho_{min})} \int_{\rho_{min}}^{\rho_{max}} \frac{\rho_{max} - \rho}{\rho_{max} + 1} d\rho \qquad (3.14)$$

$$\bar{\varepsilon}(\text{Null},\mu) = 1 - \frac{1}{(\rho_{max} - \rho_{min})} \frac{(\rho_{min} - \rho_{max})^2}{2(1 + \rho_{max})} = 1 - \frac{\rho_{max} - \rho_{min}}{2(1 + \rho_{max})} \qquad (3.15)$$

Figure 3.4 graphically depicts the average constant service rate efficiency. The point highlighted is the average across the range [0.1,0.9] which was depicted in Figure 3.2 and with an average efficiency of 0.789.

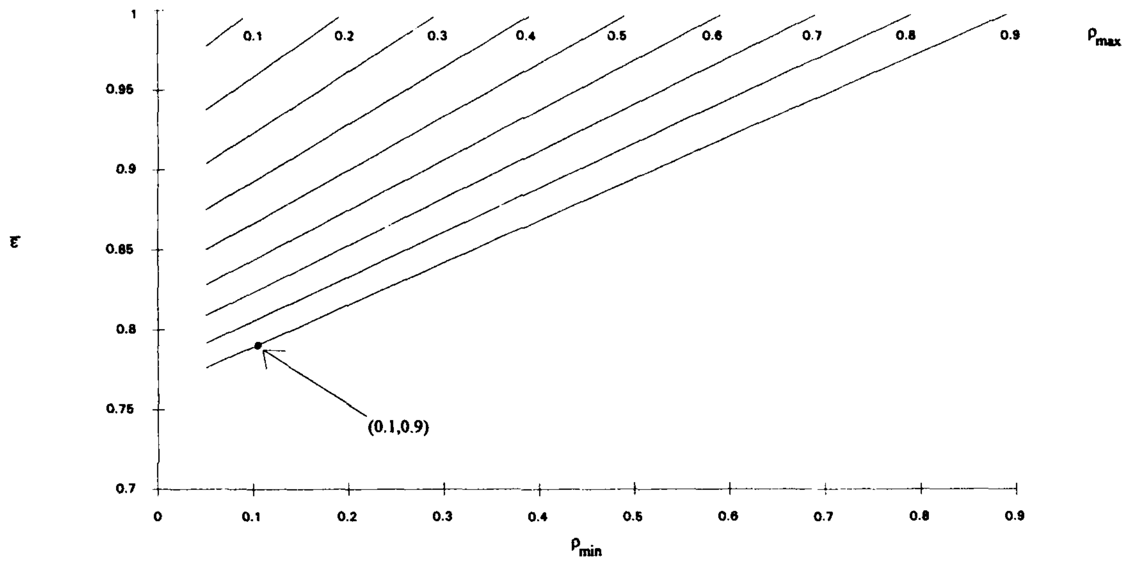


Figure 3.4 - Average Efficiency, Service Rate Constant.

3.1.7 Processor Efficiency - Back End Hybrid.

No integration is required for this case as it operates at 100% efficiency across all processors.

3.1.8 Processor Efficiency - Front End Hybrid.

The same integration to compute the average processor efficiency can also be performed for the front end hybrid scheme using Equation 3.10.

$$\bar{\varepsilon}(\text{Null}, \text{FE Hybrid}) = 1 - \frac{1}{(\rho_{max} - \rho_{min})} \int_{\rho_{min}}^{\rho_{max}} \frac{\rho_{max} - \rho_{min}}{1 + 2\rho_{max} + \rho_{max}\rho_{min}} d\rho \qquad (3.16)$$

Solving the integration yields:

$$\bar{\varepsilon}(\text{Null, FE Hybrid}) = 1 - \frac{\rho_{max} - \rho_{min}}{1 + 2\rho_{max} + \rho_{max}\rho_{min}} \tag{3.17}$$

Figure 3.5 shows where the range of ρ of [0.1,0.9] would plot with an average efficiency of 0.723.

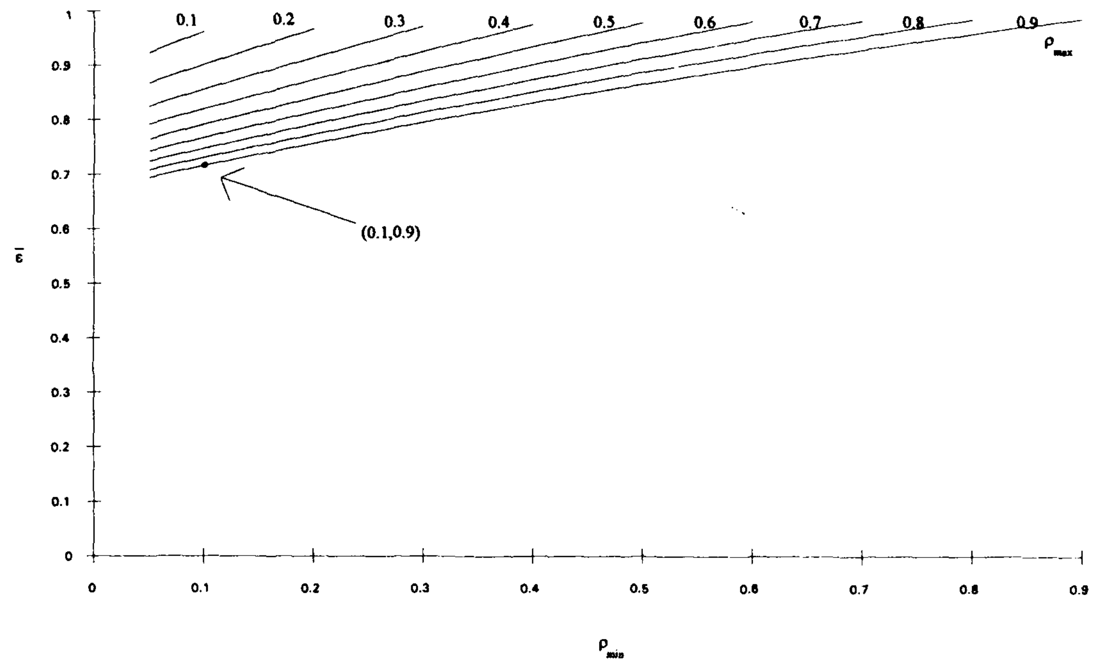


Figure 3.5 - Average Efficiency, Front End Hybrid.

The best choice would be to use the back end hybrid case which would provide 100% efficient use of all of the processors. There may be some cases however, when it may be more prudent to hold either λ or μ constant to study the effect on the system by varying only one parameter. Comparison of Figures 3.3 and 3.4 lead to the conclusion that holding μ constant will lead to more efficient overall parallel simulations than holding λ constant. Holding μ constant for $\rho = [0.1,0.9]$ has an efficiency of 0.789 while holding λ constant has only 0.341.

3.2 Simulation Efficiency - Null Events and The State 0 Effect.

Another potential drawback of the time synchronous/ standard clock algorithm is observed during the state update portion of the parallel simulations. For the M/M/1 queue, the algorithm will attempt generation of a departure event regardless of the current system state. From an efficiency standpoint, this leads to more wasted computational effort. Figure 3.6 depicts the effective event generation rates, both actual and wasted, at processor i. Note that the steady state probability of system i being in any state other than 0 is ρ_i.

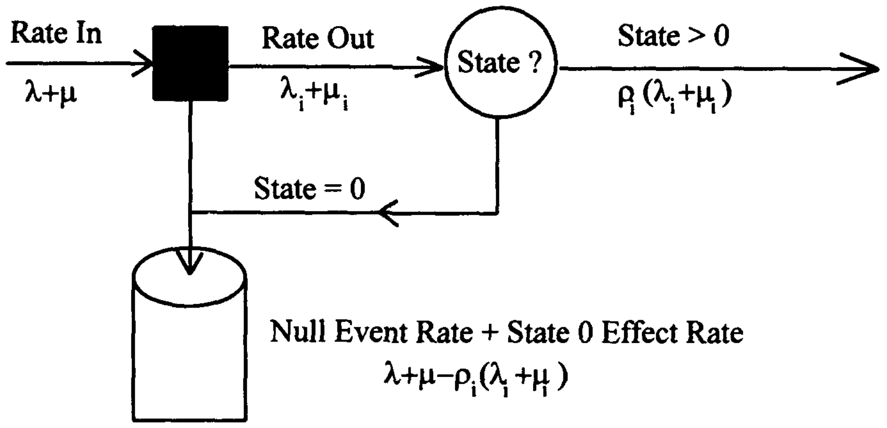

Rate In
$\lambda+\mu$
Rate Out
$\lambda_i+\mu_i$
State ?
State > 0
$\rho_i(\lambda_i+\mu_i)$
State = 0
Null Event Rate + State 0 Effect Rate
$\lambda+\mu-\rho_i(\lambda_i+\mu_i)$

Figure 3.6 - Null Event Splitting/ Thinning and State 0 Effect.

Equation 3.18 provides the probability formulation of an inefficient event which includes both a Null event and an *attempted departure* from state 0.

$$P(\text{Null}_0) = P(\text{Departure} \cap \text{State0} \cap \overline{\text{Null}}) + P(\text{Null}) \tag{3.18}$$

$P(\text{Departure} \cap \text{State0} \cap \overline{\text{Null}}) = P(\text{Departure}) * P(\text{State } 0)$ because either the generation of a departure event or the system being in state 0 are independent. Also, a Null event can still be generated while the system is in state 0 and is accounted for by the second term of Equation 3.18.

3.2.1 Processor Efficiency - Arrival Rate Constant.

Once again, the case of holding λ constant will be examined. The first term of the next equation signifies the probability of a departure event while the system is in state 0. The second term is the Null event probability.

$$P(\text{Null}_0,\lambda)\big|_i = \left(\frac{\mu_i}{\lambda+\mu_{max}}\right)(1-\rho_i) + \frac{\mu_{max}-\mu_i}{\lambda+\mu_{max}} \tag{3.19}$$

Algebraically solving equation 3.19 using the same relationships between $\lambda, \mu_{max}, \rho_i$ and ρ_{min} that were used in the previous section, and solving in terms of ρ yields:

$$P(\text{Null}_0,\lambda)\big|_i = \frac{1-\rho_{min}}{1+\rho_{min}} \tag{3.20}$$

Once again, the important aspect of efficiency is the overall average across all the processors, so integrating across the possible range of utilization ratios yields the following equations.

$$\bar{\varepsilon}(\text{Null}_0,\lambda) = 1 - \frac{1}{(\rho_{max}-\rho_{min})}\int_{\rho_{min}}^{\rho_{max}} \frac{1-\rho_{min}}{1+\rho_{min}} d\rho \tag{3.21}$$

Evaluating Equation 3.21 yields:

$$\bar{\varepsilon}(\text{Null}_0,\lambda) = 1 - \frac{1-\rho_{\text{min}}}{1+\rho_{\text{min}}} = \frac{2\rho_{\text{min}}}{1+\rho_{\text{min}}} \tag{3.22}$$

Figure 3.7 provides a graphical representation of Equation 3.22 with the range point of [0.1,0.9] once again highlighted.

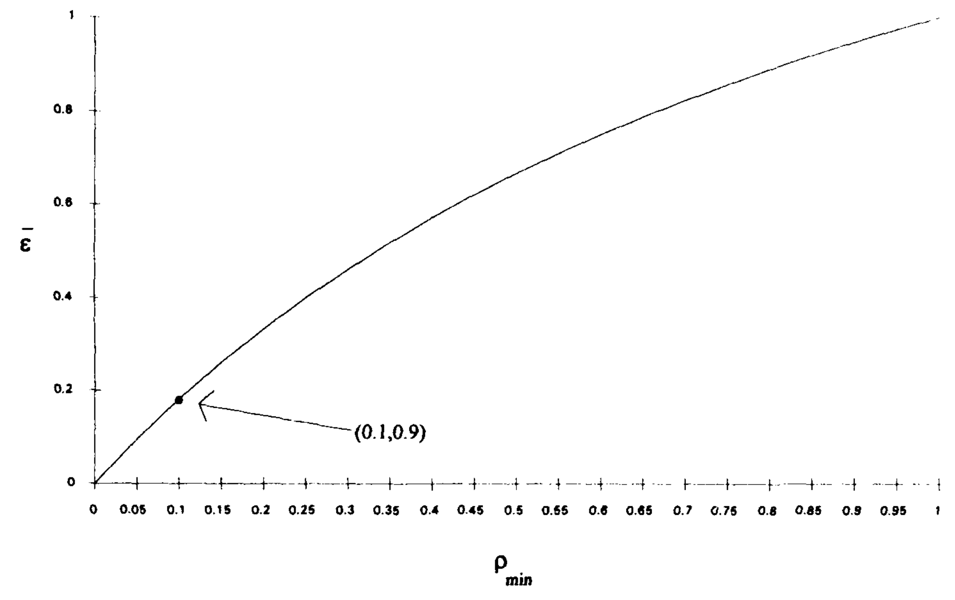


Figure 3.7 - Average Efficiency, Arrival Rate Constant, with State 0 Effect.

The point highlighted has an average efficiency, $\bar{\varepsilon}$=0.182. Note that ρ_{max} is not a factor. With the state 0 effect not included, $\bar{\varepsilon}$=0.341. Therefore, an average of 15.9% of the generated events will be attempted departures when any system is in state 0 for this range of ρ.

3.2.2 Processor Efficiency - Service Rate Constant.

Similar computations can also be accomplished for the μ constant case. Once again the probability of a departure given the system is in state 0 is added to the Null event probability determine its effect.

$$P(\mathrm{Null}_0,\mu)\big|_i = \left(\frac{\mu}{\lambda_{max}+\mu}\right)(1-\rho_i)+\frac{\lambda_{max}-\lambda_i}{\lambda_{max}+\mu} \tag{3.23}$$

Substituting and rearranging Equation 3.23 in terms of ρ:

$$P(\mathrm{Null}_0,\mu)\big|_i = \frac{1-2\rho_i+\rho_{max}}{1+\rho_{max}} \tag{3.24}$$

Applying the same limits of integration to Equation 3.24 results in the following equation. The integral is then evaluated to compute the average efficiency.

$$\bar{\varepsilon}(\mathrm{Null}_0,\mu) = 1-\frac{1}{(\rho_{max}-\rho_{min})}\int_{\rho_{min}}^{\rho_{max}}\frac{1-2\rho+\rho_{max}}{1+\rho_{max}}d\rho = \frac{\rho_{max}+\rho_{min}}{1+\rho_{max}} \tag{3.25}$$

Figure 3.8 provides a graphical depiction of the average processor efficiency for the μ constant case.

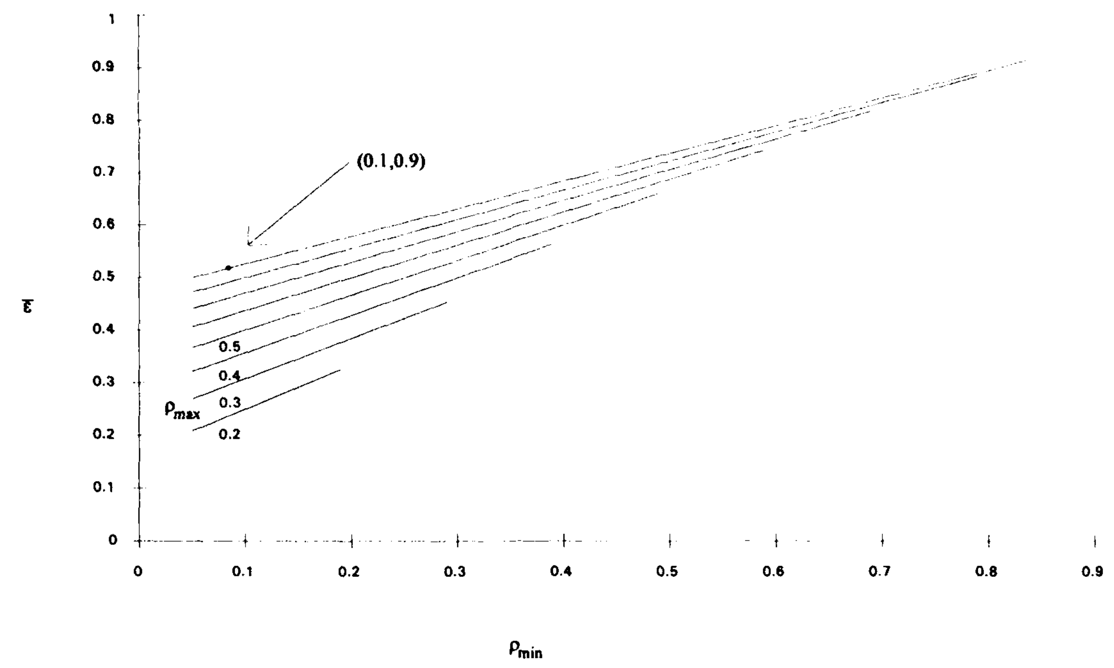


Figure 3.8 - Average Efficiency, Service Rate Constant, with State 0 Effect.

The point highlighted is the average efficiency for ρ [0.1,0.9], $\bar{\varepsilon}$=0.526. If the state 0 effect were not included, it was shown in the previous section that the average processor efficiency was 0.789. Therefore, inclusion of the state 0 effect for this range of systems of the μ constant scheme results in a 26.3% drop in efficiency.

3.2.3 Processor Efficiency - Back End Hybrid.

The case in which both the arrival and service rates are varied and marked on the back end will be examined for the decrease in efficiency due to this state 0 effect. Recall that P(Null)=0 for this method. Therefore, the only inefficient events are the state 0 attempted departures.

$$P(\text{Null}_0, \text{BE Hybrid})\big|_i = \left(\frac{\mu_i}{\lambda_i + \mu_i}\right)(1 - \rho_i) \tag{3.26}$$

By substitution and subsequent factoring out of the μ_i's, Equation 3.26 can be expressed in terms of ρ.

$$P(\text{Null}_0, \text{BE Hybrid})\Big|_i = \frac{1-\rho_i}{1+\rho_i} \tag{3.27}$$

While this appears to be a simple equation, integrating across the processors yields a complicated formula for determining the average efficiency for this method.

$$\bar{\varepsilon}(\text{Null}_0, \text{BE Hybrid}) = 1 - \frac{1}{(\rho_{max} - \rho_{min})} \int_{\rho_{min}}^{\rho_{max}} \frac{1-\rho}{1+\rho} d\rho$$

$$\bar{\varepsilon}(Null_0, BE\ Hybrid) = 2\left(\frac{\rho_{max} - \rho_{min} + \ln(\frac{1+\rho_{max}}{1+\rho_{min}})}{(\rho_{max} - \rho_{min})} \right) \tag{3.28}$$

Equation 3.28 is plotted for a representative range of ρ's in Figure 3.9.

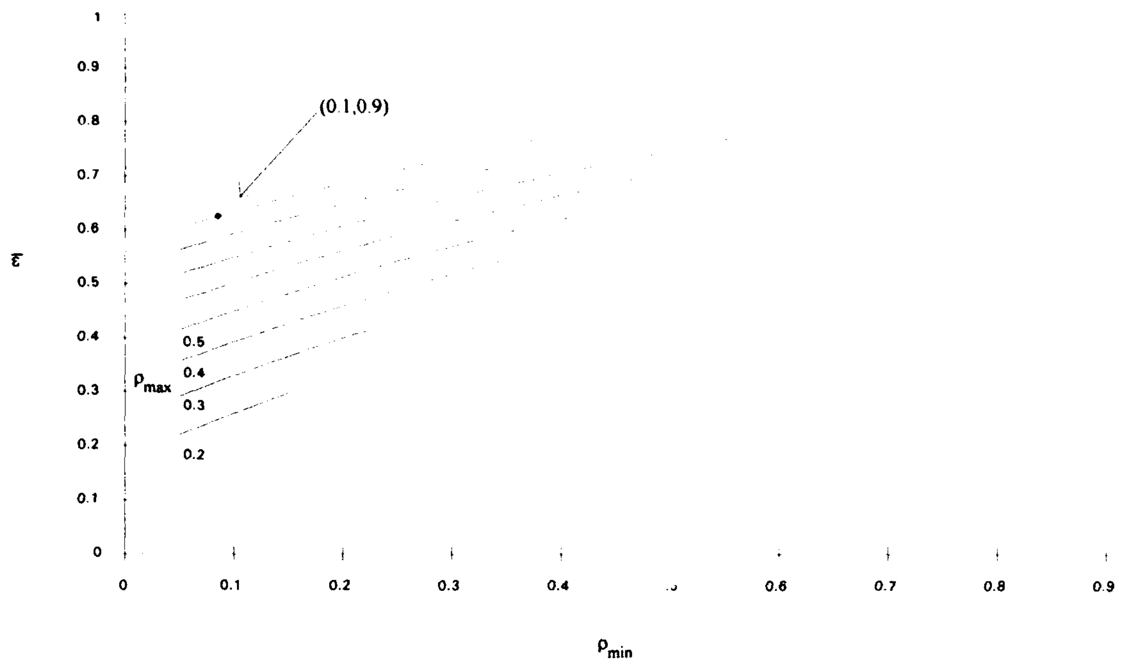


Figure 3.9 - Average Efficiency, Back End Hybrid, with State 0 Effect.

The point highlighted is the average efficiency for ρ [0.1,0.9], $\bar{\varepsilon}$=0.634. Therefore for the back end hybrid method, an average of 36.6% of all generated events will be attempted departures while a system is in state 0.

3.2.4 Processor Efficiency - Front End Hybrid.

The last case that will be considered is the front end hybrid. Once again, the probability of a departure given a system is in state 0 is combined with the probability of a null event.

$$P(\text{Null}_0,\text{FE Hybrid})\Big|_i = \left(\frac{\mu_i}{\lambda_{max}+\mu_{max}}\right)(1-\rho_i)+\frac{\lambda_{max}+\mu_{max}-\lambda_i-\mu_i}{\lambda_{max}+\mu_{max}} \tag{3.29}$$

Using similar techniques as in the other methods, Equation 3.29 can be rearranged in terms of ρ:

$$P(\text{Null}_0, \text{FE Hybrid})\Big|_i = 1 - \frac{2\rho_i}{(1+\rho_i)} \frac{(1+\rho_{\max})(1+\rho_{\min})}{(1+2\rho_{\max}+\rho_{\max}\rho_{\min})} \tag{3.30}$$

Average processor efficiency is computed in the same way by integrating across the range of interest.

$$\bar{\varepsilon}(\text{Null}_0, \text{FE Hybrid}) = 1 - \frac{1}{(\rho_{\max}-\rho_{\min})} \int_{\rho_{\min}}^{\rho_{\max}} 1 - \frac{2\rho_i}{(1+\rho_i)} \frac{(1+\rho_{\max})(1+\rho_{\min})}{(1+2\rho_{\max}+\rho_{\max}\rho_{\min})} d\rho$$

$$\bar{\varepsilon}(\text{Null}_0, \text{FE Hybrid}) = \frac{2(\rho_{\max}-\rho_{\min}+\ln(\frac{1+\rho_{\min}}{1+\rho_{\max}}))(1+\rho_{\max})(1+\rho_{\min})}{(\rho_{\max}-\rho_{\min})(1+2\rho_{\max}+\rho_{\max}\rho_{\min})} \tag{3.31}$$

Figure 3.10 provides the graphical representation of Equation 3.31. Note the similarity between the two hybrid methods. The front end appears to have lower overall average efficiency across the entire possible range of systems.

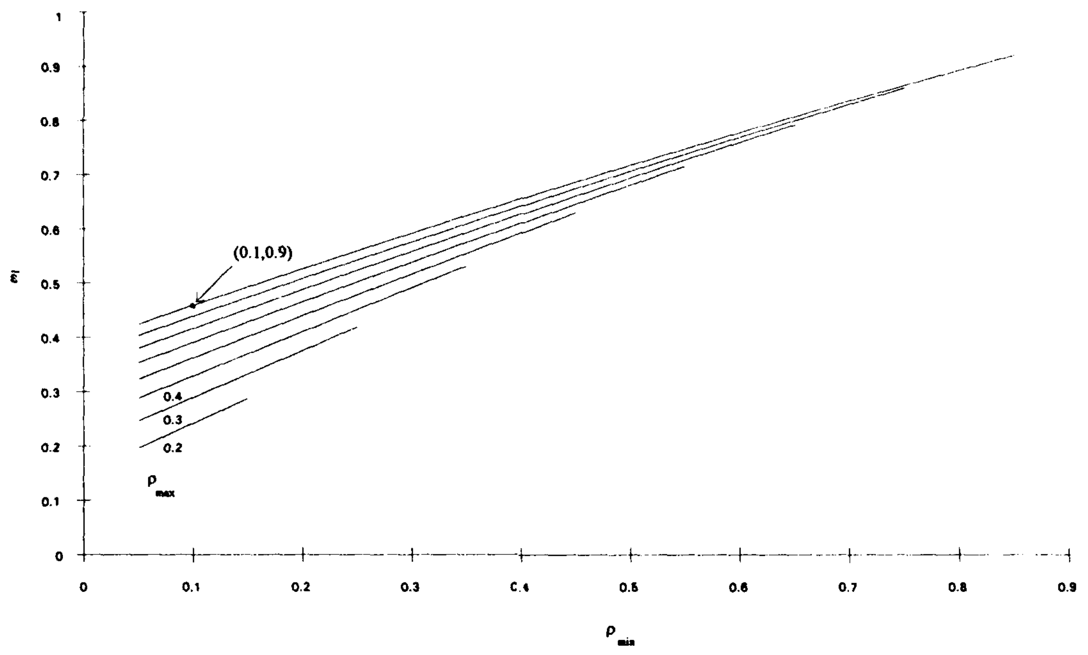


Figure 3.10 - Average Efficiency, Front End Hybrid, with State 0 Effect.

The highlighted point in Figure 3.9 depicts an average efficiency of 0.458 for the same range that the other systems were computed at.

Comparing all three methods, the back end hybrid case provides the best average efficiency for the M/M/1 queue.

ρ	λ Constant	μ Constant	Front End Hybrid	Back End Hybrid
[0.1,0.9]	0.183	0.526	0.458	**0.634**

Table 3.2 - M/M/1 Efficiency Results.

3.3 Simulation Efficiency - Null Events and State 0,k Effects.

The efficiency concepts developed in the preceding sections will now be extended to the M/M/1/k queueing system. An additional inefficient event during any clock cycle for this system is an *attempted arrival* while the system is in state k. Adding this possibility to the sum of the inefficient events generated from both Null and state 0 effects is described by Equation 3.32.

$$P(\text{Null}_{0,k}) = P(\text{Null}_0) + P(\text{Arrival} \cap \text{State } k \cap \overline{\text{Null}}) \tag{3.32}$$

For an M/M/1/k queue, the long run probability of being in any state is:

$$\begin{aligned} P(\text{State } j) &= \frac{1-\rho}{1-\rho^{k+1}}\rho^j, \rho \neq 1 \\ P(\text{State } j) &= \frac{1}{1+k}, \rho = 1 \end{aligned} \tag{3.33}$$

The long run probabilities of being in any state are different than those for the M/M/1 so each inefficient type must be included in the new set of equations rather than just incrementally adding the state k effects. Figure 3.11 now depicts the resulting actual and inefficient rates of this queueing system at processor i.

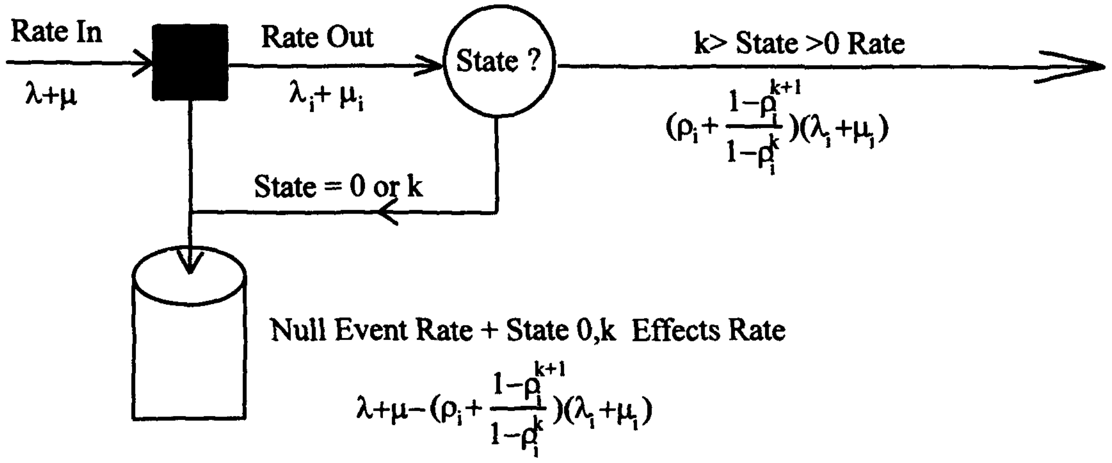

Rate In
$\lambda+\mu$
Rate Out
$\lambda_i+\mu_i$
State ?
k> State >0 Rate
$(\rho_i+\frac{1-\rho_i^{k+1}}{1-\rho_i^k})(\lambda_i+\mu_i)$
State = 0 or k
Null Event Rate + State 0,k Effects Rate
$\lambda+\mu-(\rho_i+\frac{1-\rho_i^{k+1}}{1-\rho_i^k})(\lambda_i+\mu_i)$

Figure 3.11 - Null Event Splitting/ Thinning and State 0,k Effects.

3.3.1 Processor Efficiency - Arrival Rate Constant.

The case of holding the arrival rates constant and varying the service rates across the processors has been previously shown to be the least efficient method. Two additional steps must be taken to accurately compute efficiency. The first step is adding the attempted departure while in state k probability to the previously analyzed effects. Secondly, a separate equation is now required when ρ=1.0, otherwise division by 0 would occur. At a utilization of 1.0, the probability of being in any state is equally likely (Equation 3.33). The following sets of equations depict the inefficient event probability.

$$P(\text{Null}_{0,k},\lambda)\Big|_i = \left(\frac{\mu_i}{\lambda+\mu_{max}}\right)\left(\frac{1-\rho_i}{1-\rho_i^{k+1}}\right)+\frac{\mu_{max}-\mu_i}{\lambda+\mu_{max}}+\left(\frac{\lambda}{\lambda+\mu_{max}}\right)\left(\frac{1-\rho_i}{1-\rho_i^{k+1}}\right)(\rho_i^k),\rho_i \neq 1$$

$$P(\text{Null}_{0,k},\lambda)\Big|_i = \frac{1-\rho_{min}(1-2\rho_i^k+\rho_i^{k+1})-\rho_i^{k+1}}{(1+\rho_{min})(1-\rho_i^{k+1})},\rho_i \neq 1 \qquad (3.34)$$

$$P(\text{Null}_{0,k},\lambda)\Big|_i = \left(\frac{\mu_i}{\lambda+\mu_{max}}\right)\left(\frac{1}{1+k}\right)+\frac{\mu_{max}-\mu_i}{\lambda+\mu_{max}}+\left(\frac{\lambda}{\lambda+\mu_{max}}\right)\left(\frac{1}{1+k}\right),\rho_i = 1$$

$$P(\text{Null}_{0,k},\lambda)\Big|_i = \frac{1+k+\rho_{min}(1-k)}{1+k+\rho_{min}(1+k)},\rho_i = 1 \qquad (3.35)$$

Consider now that two parametric variations can be generated across the available processors. Both ρ and k can be varied along separate axes forming a processor grid. Since there are now two parametric variations across the processors, each processor in the grid will have an individual efficiency. Figure 3.12 depicts how efficient each of 800 processors would be using this event generation method over the range of 40 variants of ρ =[0.1,0.9] and a range of k=[1,20].

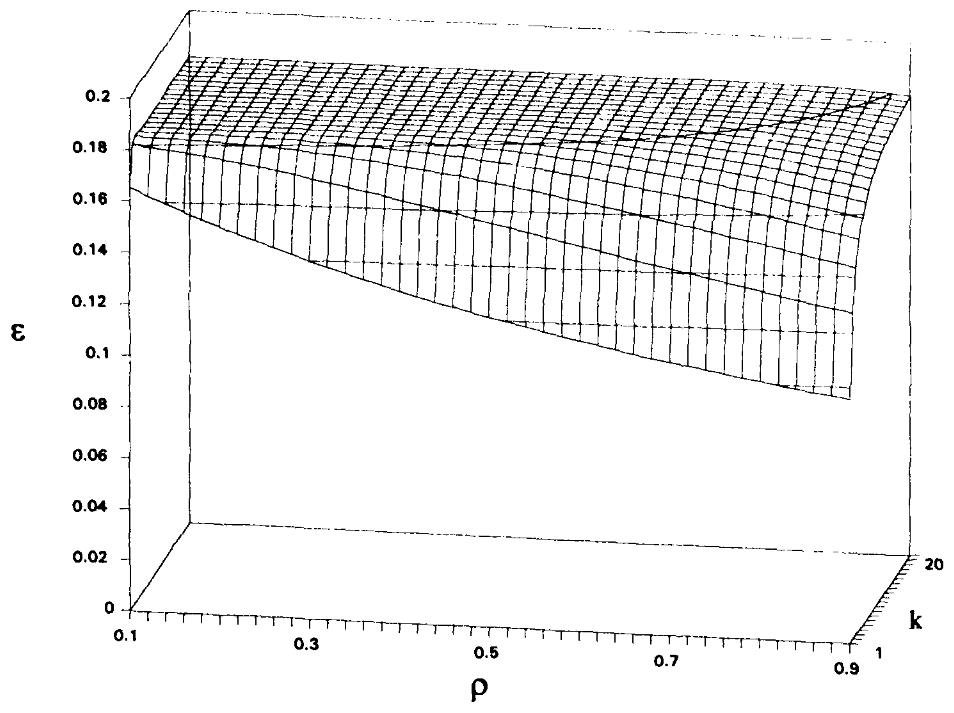


Figure 3.12 - Individual Processor Efficiency, Null and States 0,k Effects - Arrival Rate Constant.

Solving for the average efficiency is slightly different. As before, the individual processor efficiency could be integrated with respect to ρ over the range of processors, but now must be averaged conventionally across the range of buffer variants as the buffer size is a discrete quantity. When the range of ρ encompasses 1, then Equation 3.35 must be incorporated. The average efficiency then combines both a summation and an integration in order to be computed. $\frac{1}{k_{max}}\sum_{k=1}^{k_{max}}(.)$ provides the buffer component input to the average efficiency and $\frac{1}{(\rho_{max}-\rho_{min})}\int_{\rho_{min}}^{\rho_{max}}(.)$ provides the utilization ratio input as before.

$$\bar{\varepsilon}(\text{Null}_{0,k},\lambda)=1-(\frac{1}{k_{max}})(\frac{1}{\rho_{max}-\rho_{min}})\sum_{k=1}^{k_{max}}\int_{\rho_{min}}^{\rho_{max}}\frac{1-\rho_{min}(1-2\rho^{k}+\rho^{k+1})-\rho^{k+1}}{(1+\rho_{min})(1-\rho^{k+1})}d\rho,\rho\neq 1 \qquad (3.36)$$

$$\bar{\varepsilon}(\mathrm{Null}_{0,k},\lambda) = 1 - \left(\frac{1}{k_{max}}\right)\left(\frac{1}{\rho_{max} - \rho_{min}}\right)\sum_{k=1}^{k_{max}}\int_{\rho_{min}}^{\rho_{max}} \frac{1 + k + \rho_{min}(1-k)}{1 + k + \rho_{min}(1+k)} d\rho, \rho = 1$$

$$\bar{\varepsilon}(\mathrm{Null}_{0,k},\lambda) = 1 - \left(\frac{1}{k_{max}}\right)\sum_{k=1}^{k_{max}} \frac{1 + k + \rho_{min}(1-k)}{1 + k + \rho_{min}(1+k)}, \rho = 1 \qquad (3.37)$$

Equation 3.36 does not have a closed form solution for the integral however a closed form could be computed for the case where ρ=1.0. A numerical approximation was developed and is depicted in Figure 3.13. k_{max}=20 for this and all subsequent computations in this section. $\bar{\varepsilon}$ does not change significantly as k_{max} is increased above 20. Figure 3.13 has been plotted using a log scale to show the that the relative change in average efficiency is small as ρ_{min} increases above 1.0.

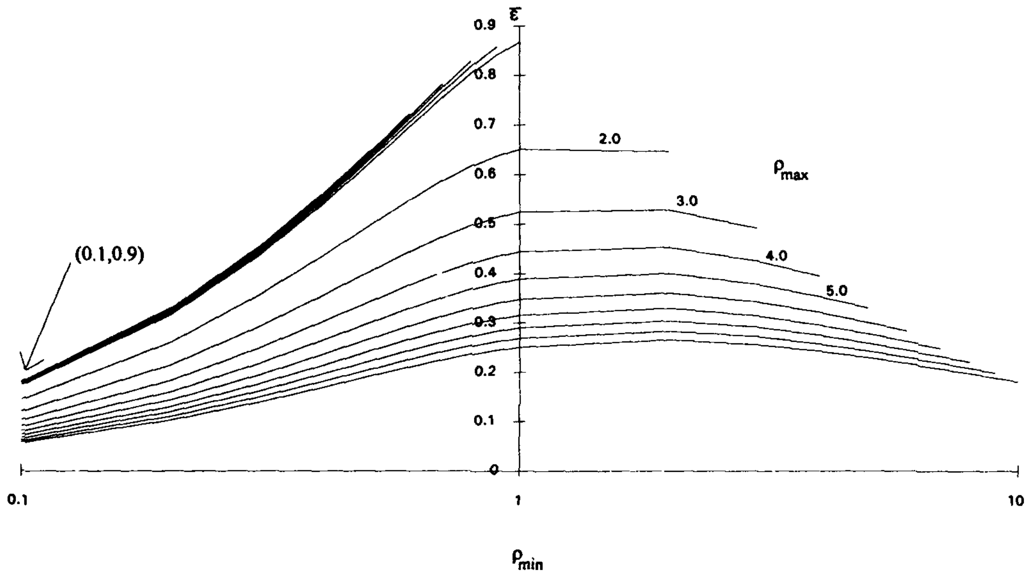


Figure 3.13 - Average Processor Efficiency, Null and States 0,k Effects - Arrival Rate Constant.

The point identified is where Figure 3.12 with ranges of ρ=[0.1,0.9] and k=[1,20] plots with an average efficiency $\bar{\varepsilon}$= 0.175. Comparing the efficiency results for the three effects shows the relatively minor decrease in efficiency when that state k effect is included.

Null Events	w/ State 0	w/ State 0,k
0.341	0.182	0.175

Table 3.3 - M/M/1/k Efficiency Comparison - Arrival Rate Constant.

3.3.2 Processor Efficiency - Service Rate Constant.

Attempted arrivals while the system is in state k are now added to the efficiency computations of the service rate constant method. Once again, a separate equation is required for the situation where ρ=1.

$$P(Null_{0,k},\mu)\big|_i = (\frac{\mu}{\lambda_{max}+\mu})(\frac{1-\rho_i}{1-\rho_i^{k+1}}) + \frac{\lambda_{max}-\lambda_i}{\lambda_{max}+\mu} + (\frac{\lambda_i}{\lambda_{max}+\mu})(\frac{1-\rho_i}{1-\rho_i^{k+1}})(\rho_i^k), \rho_i \neq 1$$

$$P(Null_{0,k},\mu)\big|_i = (\frac{\mu}{\lambda_{max}+\mu})(\frac{1}{1+k}) + \frac{\lambda_{max}-\lambda_i}{\lambda+\mu_{max}} + (\frac{\lambda_i}{\lambda_{max}+\mu})(\frac{1}{1+k}), \rho_i = 1$$

The preceding two equations are now rearranged by using the same type of substitutions for λ and μ to allow them to be rearranged in terms of ρ.

$$P(Null_{0,k},\mu)\big|_i = \frac{1-2\rho_i+\rho_{max}+\rho_i^{k+1}(1-\rho_{max})}{(1+\rho_{max})(1-\rho_i^{k+1})}, \rho_i \neq 1 \tag{3.38}$$

$$P(Null_{0,k},\mu)\big|_i = \frac{1-k+\rho_{max}(1+k)}{(1+\rho_{max})(1+k)}, \rho_i = 1 \tag{3.39}$$

Figure 3.14 depicts the 800 processors once again with their individual efficiency using the μ constant method over the range of ρ=[0.1,0.9] and k=[1,20]. There appears to be great overall efficiency improvement over the λ constant method. Also note that some processors are now generating events at a relatively high efficiency.

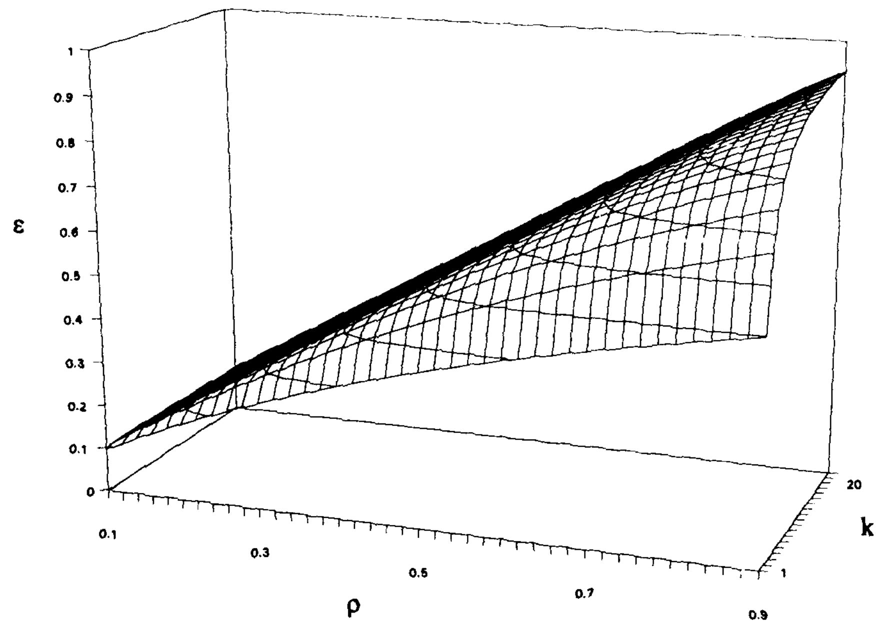


Figure 3.14 - Individual Processor Efficiency, Null and States 0,k Effects - Service Rate Constant.

Solving for average processor efficiency requires the use once again of a summation and an integration as in the λ constant method.

$$\bar{\varepsilon}(\mathrm{Null}_{0,k},\mu) = 1-\left(\frac{1}{k_{max}}\right)\left(\frac{1}{\rho_{max}-\rho_{min}}\right)\sum_{k=1}^{k_{max}}\int_{\rho_{min}}^{\rho_{max}}\frac{1-2\rho+\rho_{max}+\rho^{k+1}(1-\rho_{max})}{(1+\rho_{max})(1-\rho^{k+1})}d\rho, \rho \neq 1 \qquad (3.40)$$

$$\bar{\varepsilon}(\text{Null}_{0,k},\mu) = 1 - (\frac{1}{k_{max}})(\frac{1}{\rho_{max} - \rho_{min}})\sum_{k=1}^{k_{max}}\int_{\rho_{min}}^{\rho_{max}} \frac{1 - k + \rho_{max}(1+k)}{(1+\rho_{max})(1+k)} d\rho, \rho = 1$$

$$\bar{\varepsilon}(\text{Null}_{0,k},\mu) = 1 - (\frac{1}{k_{max}})\sum_{k=1}^{k_{max}} \frac{1 - k + \rho_{max}(1+k)}{(1+\rho_{max})(1+k)}, \rho = 1 \qquad (3.41)$$

A closed form does not exist for Equation 3.40. A numerical approximation was developed to generate Figure 3.15.

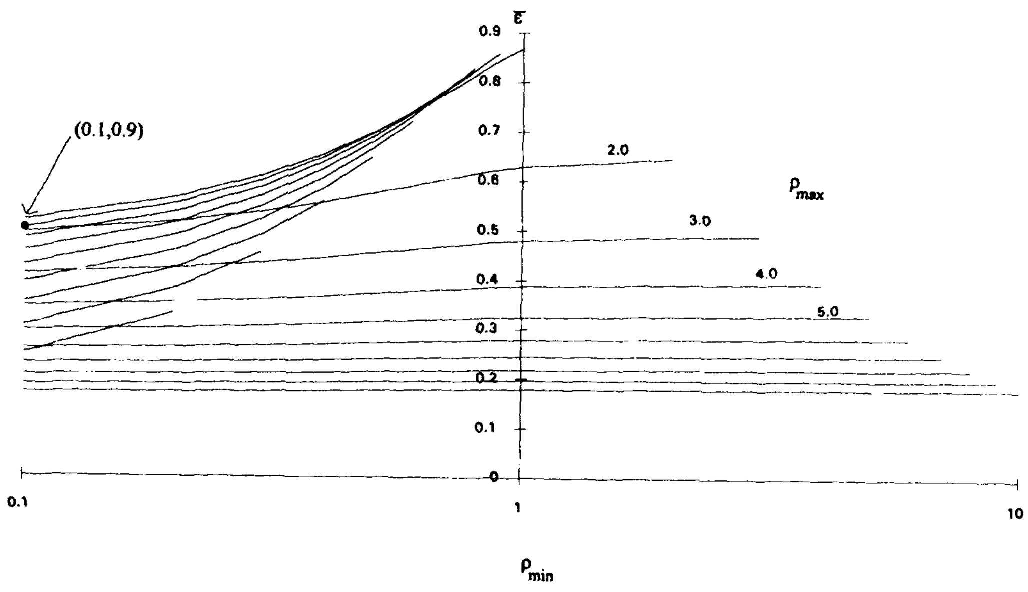


Figure 3.15 - Average Processor Efficiency, Null and States 0,k Effects - Service Rate Constant.

The average efficiency of Figure 3.14 with a range of ρ=[0.1,0.9] and k=[1,20] is plotted on Figure 3.15 with $\bar{\varepsilon}$=0.502. Once again, comparing the relative efficiency differences shows where most of the inefficiency is generated. For this scheme, the state k effect is low.

Null Events	w/ State 0	w/ State 0,k
0.789	0.526	0.502

Table 3.4 - M/M/1/k Efficiency Comparison - Service Rate Constant.

3.3.3 Processor Efficiency - Back End Hybrid.

The back end hybrid case will now have the state k effect added to its efficiency calculations. Only two terms are required in these equations as no Null events are possible.

$$P(\mathrm{Null}_{0,k}, \mathrm{BE\ Hybrid})\Big|_i = \left(\frac{\mu_i}{\lambda_i + \mu_i}\right)\left(\frac{1-\rho_i}{1-\rho_i^{k+1}}\right) + \left(\frac{\lambda_i}{\lambda_i + \mu_i}\right)\left(\frac{1-\rho_i}{1-\rho_i^{k+1}}\right)(\rho_i^k), \rho_i \neq 1$$

$$P(\mathrm{Null}_{0,k}, \mathrm{BE\ Hybrid})\Big|_i = \left(\frac{\mu_i}{\lambda_i + \mu_i}\right)\left(\frac{1}{1+k}\right) + \left(\frac{\lambda_i}{\lambda_i + \mu_i}\right)\left(\frac{1}{1+k}\right), \rho_i = 1$$

Solving the previous two equations in terms of ρ yields the following:

$$P(\mathrm{Null}_{0,k}, \mathrm{BE\ Hybrid})\Big|_i = \frac{1-\rho_i + \rho_i^{k+1} - \rho_i^{k+2}}{1+\rho_i - \rho_i^{k+1} - \rho_i^{k+2}}, \rho_i \neq 1 \qquad (3.42)$$

$$P(\mathrm{Null}_{0,k}, \mathrm{BE\ Hybrid})\Big|_i = \frac{1}{1+k}, \rho_i = 1 \qquad (3.43)$$

Once again, processor efficiency for each of the 800 processors is depicted in Figure 3.16 with a range of ρ=[0.1,0.9] and k=[1,20]. The shape of this efficiency surface is very similar to the μ constant surface, Figure 3.14, but with apparent higher processor efficiency.

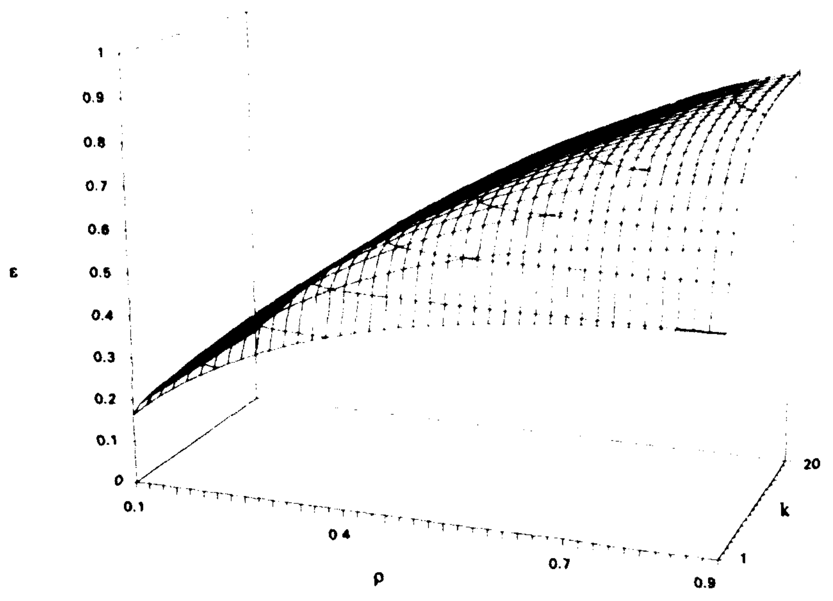


Figure 3.16 - Individual Processor Efficiency, Null and States 0,k Effects - Back End Hybrid.

Solving for average efficiency by summing over the range of k and integrating over ρ yields the following:

$$\bar{\varepsilon}(\text{Null}_{0,k},\text{BE Hybrid}) = 1-(\frac{1}{k_{max}})(\frac{1}{\rho_{max}-\rho_{min}})\sum_{k=1}^{k_{max}}\int_{\rho_{min}}^{\rho_{max}}\frac{1-\rho+\rho^{k+1}-\rho^{k+2}}{1+\rho-\rho^{k+1}-\rho^{k+2}}d\rho, \rho \neq 1 \quad (3.44)$$

$$\bar{\varepsilon}(\text{Null}_{0,k},\text{BE Hybrid}) = 1-(\frac{1}{k_{max}})(\frac{1}{\rho_{max}-\rho_{min}})\sum_{k=1}^{k_{max}}\int_{\rho_{min}}^{\rho_{max}}\frac{1}{1+k}d\rho, \rho = 1$$

$$\bar{\varepsilon}(\text{Null}_{0,k},\text{BE Hybrid}) = 1-(\frac{1}{k_{max}})\sum_{k=1}^{k_{max}}\frac{1}{1+k}, \rho = 1 \quad (3.45)$$

A closed form solution of the integral does not exist for Equation 3.44 so a numerical approximation was developed to produce the curves for average efficiency given in Figure 3.17.

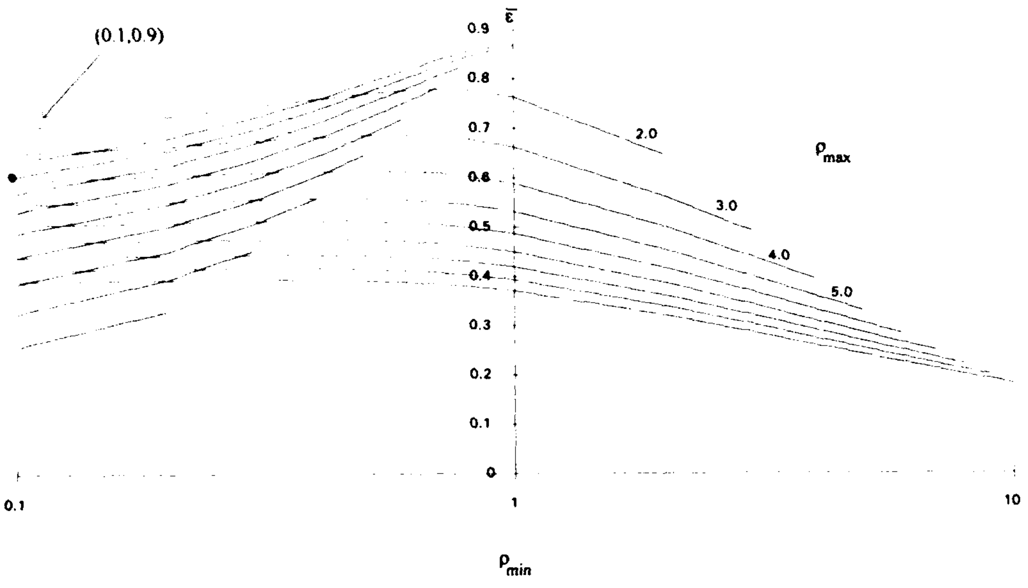


Figure 3.17 - Average Processor Efficiency Difference, Null and States 0,k Effects - Back End Hybrid.

Note that there is more of a drop-off as ρ_{min} increases above 1.0 as compared to the μ constant method. The point highlighted in this figure depicts the average efficiency of $\rho = [0.1,0.9]$,k=[1,20] with an average efficiency, $\bar{\varepsilon} = 0.605$.

Null Events	w/ State 0	w/ State 0,k
1.000	0.634	0.605

Table 3.5 - M/M/1/k Efficiency Comparison - Back End Hybrid.

This method provides the most efficient M/M/1/k range of systems thus far.

3.3.4 Processor Efficiency - Front End Hybrid.

The efficiency of the hybrid case where the marked Poisson process is generated on the front end with the state k effect added follows. The probability of a Null event is equivalent to the probability of not accepting the event when it reaches the back end and is the second term in the following two equations.

$$P(\text{Null}_{0,k},\text{FE Hybrid})\Big|_i = \left(\frac{\mu_i}{\lambda_{max}+\mu_{max}}\right)\left(\frac{1-\rho_i}{1-\rho_i^{k+1}}\right) + \frac{\lambda_{max}+\mu_{max}-\lambda_i-\mu_i}{\lambda_{max}+\mu_{max}} + \left(\frac{\lambda_i}{\lambda_i+\mu_i}\right)\left(\frac{1-\rho_i}{1-\rho_i^{k+1}}\right)(\rho_i^k), \rho_i \neq 1$$

$$P(\text{Null}_{0,k},\text{FE Hybrid})\Big|_i = \left(\frac{\mu_i}{\lambda_{max}+\mu_{max}}\right)\left(\frac{1}{1+k}\right) + \frac{\lambda_{max}+\mu_{max}-\lambda_i-\mu_i}{\lambda_{max}+\mu_{max}} + \left(\frac{\lambda_i}{\lambda_{max}+\mu_{max}}\right)\left(\frac{1}{1+k}\right), \rho_i = 1$$

Solving each of the previous two equations in terms of ρ gives Equations 3.46 and 3.47.

$$P(\text{Null}_{0,k},\text{FE Hybrid})\Big|_i = 1 + \frac{2\rho_i^{k+1}-2\rho_i}{(1-\rho_i^{k+1})(1+\rho_i)} \frac{(1+\rho_{max})(1+\rho_{min})}{(1+2\rho_{max}+\rho_{max}\rho_{min})}, \rho_i \neq 1 \tag{3.46}$$

$$P(\text{Null}_{0,k},\text{FE Hybrid})\Big|_i = 1 - \frac{k(1+\rho_{max})(1+\rho_{min})}{(1+k)(1+2\rho_{max}+\rho_{max}\rho_{min})}, \rho_i = 1 \tag{3.47}$$

Figure 3.18 provides a graphical representation of the individual processor efficiency represented by Equations 3.46 and 3.47.

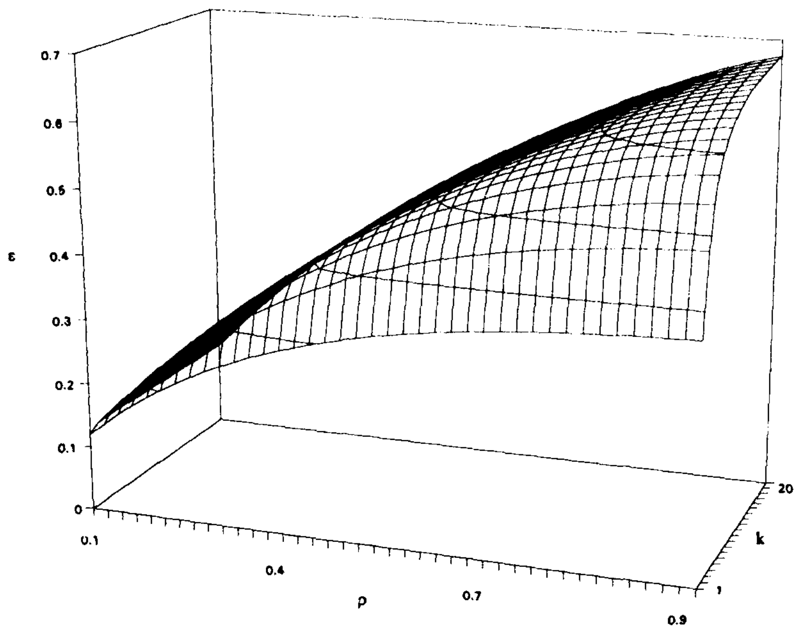


Figure 3.18 - Individual Processor Efficiency, Null and States 0,k Effects - Front End Hybrid.

Note that the shape of this surface is very similar to Figure 3.16, the back end hybrid, but with obvious lower efficiency. Summing and integrating provides the analytical formulation of the average processor efficiency.

$$\bar{\varepsilon}(\mathrm{Null}_{0,k},\mathrm{FE\ Hybrid}) = 1-(\frac{1}{k_{max}})(\frac{1}{\rho_{max}-\rho_{min}})\sum_{k=1}^{k_{max}}\int_{\rho_{min}}^{\rho_{max}} 1+\frac{2\rho_i^{k+1}-2\rho_i}{(1-\rho_i^{k+1})(1+\rho_i)}\frac{(1+\rho_{max})(1+\rho_{min})}{(1+2\rho_{max}+\rho_{max}\rho_{min})}d\rho, \rho_i \neq 1$$

$$\bar{\varepsilon}(\mathrm{Null}_{0,k},\mathrm{FE\ Hybrid}) = 1-(\frac{1}{k_{max}})(\frac{1}{\rho_{max}-\rho_{min}})\sum_{k=1}^{k_{max}}\int_{\rho_{min}}^{\rho_{max}} \frac{k(1+\rho_{max})(1+\rho_{min})}{(1+k)(1+2\rho_{max}+\rho_{max}\rho_{min})}d\rho, \rho = 1$$

$$\bar{\varepsilon}(\mathrm{Null}_{0,k},\mathrm{FE\ Hybrid}) = 1-(\frac{1}{k_{max}})\sum_{k=1}^{k_{max}} \frac{k(1+\rho_{max})(1+\rho_{min})}{(1+k)(1+2\rho_{max}+\rho_{max}\rho_{min})}, \rho = 1 \qquad (3.48)$$

Figure 3.19 provides a graphical depiction of the average processor efficiency for the front end hybrid time synchronous method. This figure was developed using a numerical approximation technique once again.

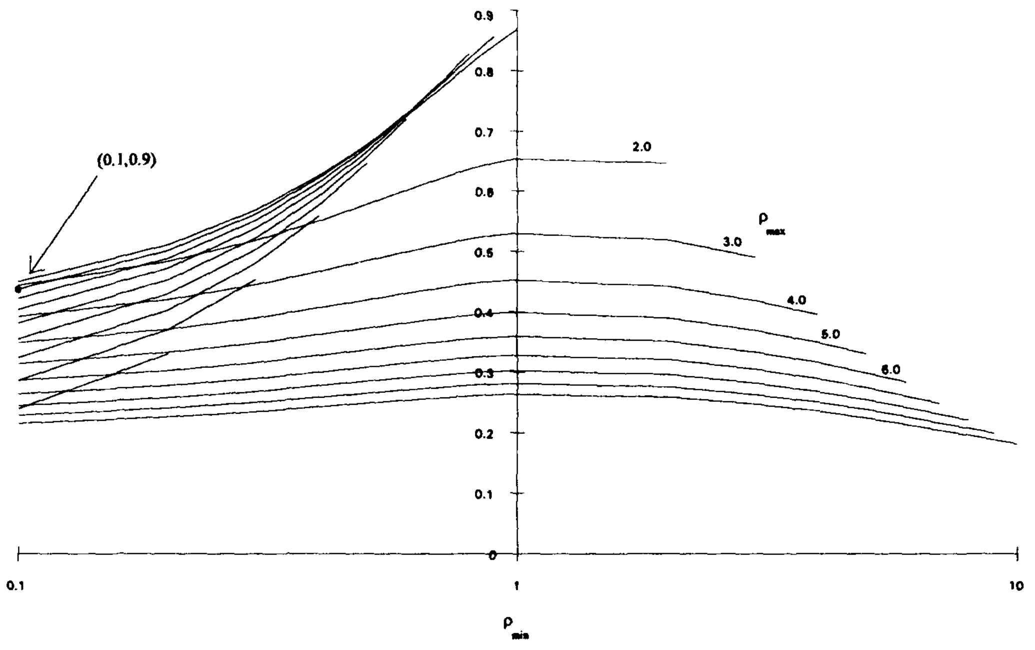


Figure 3.19 - Average Processor Efficiency, Null and States 0,k Effects - Front End Hybrid.

The point highlighted in this figure depicts the average efficiency of ρ=[0.1,0.9] ,*k*=[1,20] with an average efficiency, $\bar{\varepsilon}$ = 0.438.

Null Events	w/ State 0	w/ State 0,k
0.723	0.458	0.438

Table 3.6 - M/M/1/k Efficiency Comparison - Front End Hybrid.

3.3.5 Sources of Inefficiency - Null Events and State 0,k Effects.

Now that the overall average efficiency of the different methods of time synchronous event generation have been developed for the M/M/1/k, it is important to note the sources of the inefficiency. Consider any individual processor. At this processor, there is a unique set of system parameter values that distinguish it from all others i.e. $(\lambda,\mu,\rho,k)_i$. Three cases from each of the event generation schemes will be compared at some representative values of ρ and k.

Figure 3.20 depicts the various methods at ρ=0.5, ρ_{min}=0.1, ρ_{max}=0.9, k=1. Note the large Null event effect for λ constant and the large state 0 effect for the hybrid case.

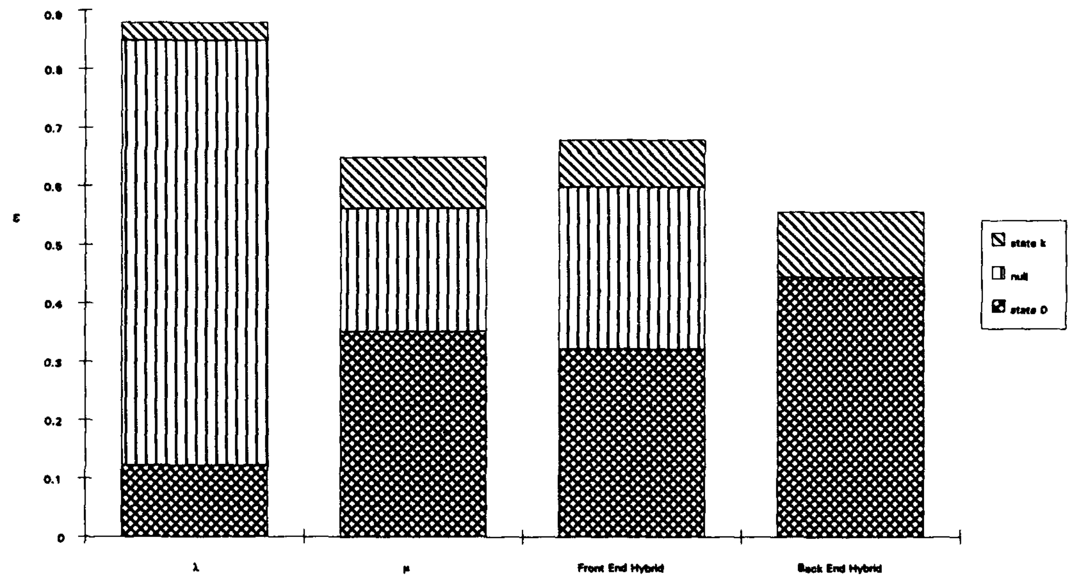


Figure 3.20 - Inefficiency Comparison, Case A.

Figure 3.21 depicts the different methods at ρ=3.0, ρ_{min}=0.1, ρ_{max}=10.0, k=1. This range was selected to demonstrate any pronounced effects if ρ=1.0 was crossed. Notice how now the state k effect dominates the hybrid case and that the Null effect still dominates the λ constant.

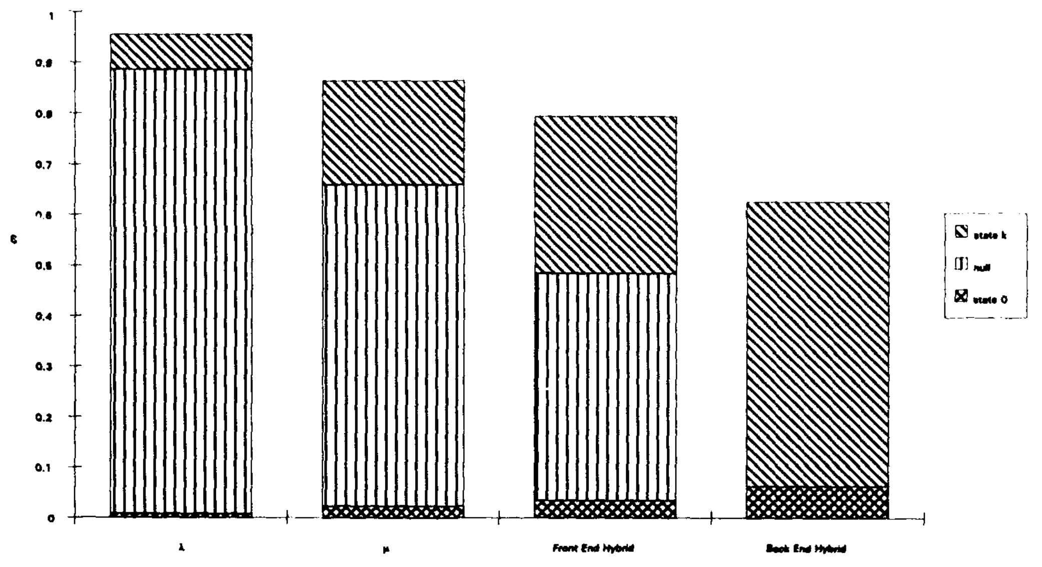


Figure 3.21 - Inefficiency Comparison, Case B.

Figure 3.22 depicts ρ=5.0, ρ_{min}=1.0, ρ_{max}=10.0, k=1. State k now dominates the hybrid as to be expected.

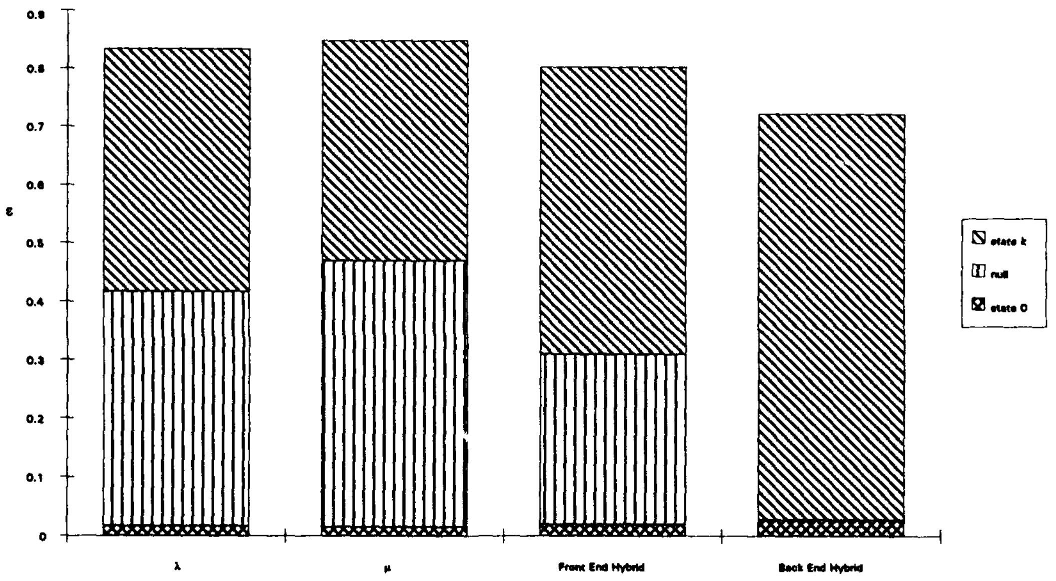


Figure 3.22 - Inefficiency Comparison, Case C.

The ρ_i's selected are roughly the midpoint of the different ranges selected. Choosing k=1 highlighted the state k effect, if any. If k were any higher, then the state k effect would diminish. The reason the ρ was selected around the mid-point was that as ρ tended toward either range extreme, the hybrid efficiency tended toward either λ or μ constant. This depended on whether the hybrid was plotted close to the minimum or maximum ρ respectively. For the ranges less than 1, the hybrid tended to lose more events proportionally due to the state 0 effect, whereas for ranges greater than 1, the state k effect is more pronounced.

3.3.6 Comparison of Efficiency - Null Events and State 0,k Effects.

Table 3.7 provides an overall comparison of the average efficiency for the three methods at the three ranges depicted in the previous section.

AVERAGE EFFICIENCY:

$[\rho_{min}, \rho_{max}]$	λ constant	μ constant	Front End Hybrid	Back End Hybrid
[0.1,0.9]	0.175	0.502	0.438	**0.605**
[0.1,10.0]	0.057	0.171	0.215	**0.392**
[1.0,10.0]	0.251	0.179	0.264	**0.371**

Table 3.7 - Efficiency Comparison.

Obviously from this table, it is definitely more efficient to use the back end hybrid generation scheme for any range of ρ.

4.0 Simulation Design.

One of the key aspects of any simulation study is the selection of the system parameters to be simulated. The CM-2 affords an opportunity to perform simultaneous multiple simulations that would otherwise take considerable computational time using serial computers. Chapter 4.0 performs parametric simulations of a fairly simple Markovian system, the M/M/1/k queue using the CM-2. This system readily parameterizes into two dimensions, the utilization factor and the buffer size. Stochastic variations can then be incorporated along a third dimension for which results can be presented in standard graphical form. Following the techniques in this chapter, more complex systems can be parameterized using additional dimensions but require more complex data visualization techniques to explore all relevant relationships about system performance.

4.1 Parameter Selection.

The M/M/1/k queue can be parameterized in several different ways. These parameters should be selected to optimize the number of required processors through the use of the compiler as well as efficiently use all available processors during the course of the simulation. Chapter 3.0 developed analytically the concept of efficiency based on the various methods of parameterizing the M/M/1 and M/M/1/k queueing systems.

The two compilation models available on the CM-2 are Paris and Slicewise. Under the Paris model, the basic processing elements are the bit-serial processors with 8K of local memory. Under the Slicewise model, the CM-2 is organized into processing nodes, each containing 32 bit-serial processors, 256K of local memory and one floating point accelerator chip. The Slicewise model uses 32 processors together to do a single 32-bit operation in one clock cycle as opposed to Paris which performs 32, 32-bit operations in 32 clock cycles. When a program is run under the Paris model at PSC, there are 32K processing elements available whereas under the Slicewise, only 1K. The Slicewise model makes use of the floating point accelerator's internal registers to perform any computations, whereas in order for the Paris model to perform any computations, it must read and write to and from memory each bit serially.

For problems which require more than the available number of processors, the CM-2 uses the notion of virtual processors. Under Paris, the number of virtual processors allocated must be a power of 2 to the number of physical processors. If the number of virtual processors is not exactly a power of 2, the compiler increases the number of virtual processors to the next power of two, resulting in lower available memory for each virtual processor. Each dimension of the VP grid must also be a power of 2. If the dimension is not a power of 2, it is rounded up to the next power, thus wasting available virtual processors and memory.

Under Slicewise, the number of virtual processors must be a multiple of 4 to the number of physical processors. This compiler also rounds up the number of virtual processors to the next multiple of 4 if not that many were initially allocated. There is no restriction on the length of any dimension as long as all total to a multiple of 4.

The most efficient M/M/1/k time synchronous parameterization scheme using utilization ratio uniformly spread across processors was for the back end hybrid method over a range of $\rho=[0.1,2.0]$ For this reason, this range was selected to compare time synchronous methods with the event synchronous. Since, the recommended minimum number of stochastic variants is 30 to get a variance estimate close to the actual true variance, then breaking up the ρ's into 0.1 step sizes and discretely incrementing k from 1 to 20, results in a total parallel simulation count of 12,000. This total conforms to the Slicewise requirement of the total dimension count being a multiple of 4.

4.2 Parallel Array Parameter Setup.

The CM-2 provides a means with which to efficiently broadcast individual data elements to the parallel processors. Once data elements are identified as parallel, there are several ways to initialize their values. Consider the data element **A.** The first step is to identify this element as a parallel array. This is done through the use of two commands. The first is similar to that used in any high level programming language: ***real, array(30,20,20):A***. The **A** array is now identified as a three dimensional array. However, the CM-2 still does not know if the array is to be serial and thus used on the front end or in parallel. Identifying it as parallel is done by several methods. ***A=0.0*** first tells the CM-2 that **A** is a parallel array and broadcasts the value of 0 to all of the local processors. The second method entails the use of a special layout command that details to the CM-2 how interprocessor communications will transpire. ***CMF$ LAYOUT A(:serial,:news,:news)*** is interpreted by the CM as follows: The first dimension is a serial dimension and hence all 30 data elements will be located at the same parallel processor. The second and third dimensions are parallel dimensions and the array will be organized such that the processors will be laid out in a two-dimensional grid with nearest neighbors indicated by indices that differ by 1. Another means of assigning individual data to the parallel processors is through the use of a ***FORALL*** command which can initialize individual elements. ***FORALL(i=1:20)A(:,:,i)=i.*** This particular command initializes all elements with the same third-dimensional index to that index, i.e., A(4,4,1)=1, A(4,7,3)=3.

The following initialization algorithm broadcasts the setup for the M/M/1/k queue. Each parallel data array is represented in **bold** and is three dimensional. The first dimension represents the stochastic variations so that all planes formed by the intersection with the z-axis contain the same initialization values.

```
rate=1.0
rho_min=0.1
rho_max=2.0
forall(i=1:20)rho(:,:,i)=(rho_max-rho_min)/19*(i-1)+rho_min
forall(i=1:20)buffer(:,i,:)=i
```

The different time synchronous methods also require setup of the arrival and service rates. The first algorithm provides the setup algorithm for the λ constant method. The back end method has λ as a parallel array because it has to be available at every parallel processor for multiple replications If only one replication is being simulated, then this element is stored on the front end. Rate is the interevent rate with which events (real or null) are actually generated.

Front End	Back End

rate = 1.0
mu_max=rate/(1+rho_min)
lambda=rate-mu_max
	rate = 1.0
	mu_max=rate/(1+rho_min)
	lambda=rate-mu_max
	mu=lambda/rho

The μ constant case is set up for both front end and back end marking in a similar fashion.

Front End	Back End

rate = 1.0
lambda_max=rate*rho_max/(1+rho_max)
mu=rate-lambda_max
	rate = 1.0
	lambda_max=rate*rho_max/(1+rho_max)
	mu=rate-lambda_max
	lambda=rho*mu

The two hybrid cases are set up in much the same way. The primary difference now is that there are two effective interevent generation rates for the front end method. There is a front end rate which is necessarily greater that the effective back end rate. This front end rate is the same as the back end rate for the λ and μ constant methods. Therefore, if more that one parameter is varied, different rates will occur.

Front End	Back End

rate = 1.0
lambda=rate*rho(1+rho)
mu=rate/(1+rho)
lambda_max=rate*rho_max/(1+rho_max)
mu_max=rate/(1+rho_min)
front_end_rate=lambda_max+mu_max
	rate = 1.0
	lambda=rate*rho/(1+rho)
	mu=rate/(1+rho)

For either method, transition probabilities must be initialized before the arrays associated with the alias method can be constructed. The front end hybrid requires two probability structures; one for the front end with which it will mark events, and one at each parallel processor with which the marked events will be accepted. Once the event is marked as an arrival or a departure on the front end, it will then be accepted at each individual processor with probability commensurate with the local system rate. Probabilities are stored on both the front and back end processors.

For all three back end time synchronous methods, the transition probabilities are initialized in the same manner. Now however, there is a third event, the Null event, with an associated probability. No front end probability computations are required.

		FRONT END MARKING:	BACK END MARKING:
Front End Actions:	Arrival: Departure:	fe_prob(1)=lambda_max/rate fe_prob(2)=mu_max/rate	N/A N/A
Back End Actions:	**Arrival:** **Departure:** **Null:**	**prob(1,:,:,:)=lambda/lambda_max** **prob(2,:,:,:)=mu/mu_max** N/A	**prob(1,:,:,:)=lambda/rate** **prob(2,:,:,:)=mu/rate** **prob(3,:,:,:)=(rate-mu-lambda)/rate**

The event synchronous probability structure is set up slightly differently. Parameterization of the local arrival and service rates could be done using any of the three schemes available. Regardless of the time synchronous parameterization, the probability structure of the event synchronous is only a function of the local utilization ratio.

Arrival:	**prob(1,:,:,:)=lambda/(lambda+mu)**
Departure:	**prob(2,:,:,:)=mu/(lambda+mu)**
Null:	**prob(3,:,:,:)=0.0**

4.3 The Parallel Alias Method.

An implied requirement of any simulation is the quick and efficient generation of the next event to perform. The Alias method is a quick way in which generate a discrete probability distribution. Since the CM-2 is an array processor architecture, this method is very efficient in determining the event type since the Alias method returns array references, The uniformized M/M/1/k system has only three total event types so the Alias method does not really provide a significant advantage. The Alias method will provide substantial computational savings in simulating larger and more complex systems as still only two instructions are required to determine the next event type.

The principle setup algorithm was taken from Bratley, Fox and Schrage [BFS, 1987]. It was necessary to adjust the algorithm presented by BFS in order for it to work properly on the SIMD architecture. Once again any step performed in parallel is highlighted in **Bold**.

n	Number of Probabilities to Alias.
p(i)	Individual Probability.
H, L	High and Low Sets Containing Array Indices.
R(i)	Adjusted Alias Probability.
A(i)	Alias Element.
θ	Error Constant for Roundoff.

SERIAL ALGORITHM	PARALLEL ALGORITHM
Initialize H and L Sets $\leftarrow \phi$	**Initialize H and L Sets $\leftarrow \phi$**
	Initialize H and L Counters = 0
Do LOOP i=1 to n	Do LOOP i=1 to n
Set R(i) $\leftarrow$ np(i)	**Set A(i) $\leftarrow$ i**
If R(i) >1, add i to H	**Set R(i) $\leftarrow$ np(i)**
	Where (R(i)-1)< θ R(i) $\leftarrow$ 1
If R(i) >1, add i to H	**If R(i) >1, add i to H; Increment H Counter**
If R(i) <1, add i to L	**If R(i) <1, add i to L; Increment L Counter**
LOOP	LOOP
If H = ϕ, STOP	Do LOOP2 i=1,n
	If i $\leq$ L & H Counters then
Select j from L and k from H	**Select j from L and k from H**
Set A(j) $\leftarrow$ k	**Set A(j) $\leftarrow$ k**
R(k) $\leftarrow$ R(k) +R(j) -1	**R(k) $\leftarrow$ R(k) +R(j) -1**
If R(k) $\leq$ 1, Remove k from H	**If R(k) $\leq$ 1, Remove k from H**
	If R(k) > 1,
	Add k to end of H;
	Increment H Counter
If R(k) < 1, Add k to L	**If R(k) <1,**
	Add k to end of L;
	Increment L Counter
Remove j from L	**Remove j from L**
Go To LOOP	**ENDIF**
	LOOP2
	STOP

A major step in setting up the back end simulations was construction of the two alias arrays in parallel so that the set of arrays is stored at each processor. Two additional steps were required to ensure the alias arrays were properly initialized. The first was the incorporation of aliasing every element to itself as the arrays are initialized. The second was the use of an error constant, θ, to see if the product of n*p was sufficiently close to 1 to preclude any roundoff errors. Both steps corrected some perceived precision problems that occurred at the local processors.

Key to the successful implementation of the parallel algorithm is the incorporation of the high and low set counters. Because so many sets of probabilities are being aliased at once, appending element indices back to the individual sets and incrementing local counters ensures that all computations are done properly and synchronously. It also provided the opportunity to determine the maximum number of steps required to alias all of the elements. While some processors will remain idle during the latter stages of the algorithm, the benefits gained through its successful implementation greatly outweigh these computational costs. During the main portion of the simulation run, only two table lookups and one comparison are required at each processor to determine the next event regardless of the number of possible events.

4.4 Statistical Computation.

The CM-2 programming languages incorporate specialized commands to facilitate global combining. Once a parallel simulation has reached a specified point during its execution, performance measure statistics are normally computed and output to files for future use and analysis.

Consider the parallel array **A**, identified as ***real, array(30,20,20) :A***. Now define **B** as ***real, array(20,20):B***. Data array **B** is to contain the arithmetic mean of the 30 data elements identified by the first index of **A** The command available to *reduce* **B**→E[**A**] is **SUM(Array[,Dim][,Mask])**. This command computes the sum of all elements of a numeric array or the sums of each of the rank-one sections parallel to the given dimension (DIM), possibly under the control of a mask. Therefore, the instruction **B=SUM(A,DIM=1)/30,** computes the arithmetic mean of **A** along its first dimension and places the resulting means in array **B**.

Variance computation is performed in a similar manner. Now consider another array **C**, which is identified with exactly the same shape as **A**. Assume **B** still contains the mean of **A** as described in the previous paragraph. The next step in computing the variance of a performance measure in parallel is to reform the three-dimensional array **C** such that it contains the mean value contained in its respective element of the two-dimensional array **B**. The following command will reform this array to contain the correct data: **forall(n=1:30) C(n,:,:)=B(:,:)**. The forall command is required in this step because of the different inherent shapes of the parallel arrays. Lastly define a fourth array **D**, which will contain the variance data. Array **D** has the same shape as **B**. Therefore, the instruction **D=SUM(((C-A)**2)/29,DIM=1)** will compute the variance in one step and store its values in array **D**. The difference was divided by 29 instead of 30 to provide an unbiased estimator of the variance. The expected value of a performance measure can be computed in one parallel instruction and variances with an additional two. The CM-2 programming languages and architecture then provide a relatively simple instruction structure with which to compute statistical performance measures in parallel.

4.5 Performance.

There are many different performance measures that characterize the performance of any queueing system. One of the advantages of choosing the M/M/1/k queueing system is that there exists analytical solutions to many of its performance measures. Since an analytical solution exists, comparison of experimental with theoretical performance will provide clear indications if the parallel simulation techniques developed thus far do work.

Mean Queue Length, $\bar{L}$, is significant in that its value can be ascertained as a function of ρ. For the M/M/1 queue, Kleinrock presents a method for computing this value [Klienrock -I, 1975]. For the following equations, P_i, is the limiting probability of being in state i. Applying this technique to the M/M/1/k queue yields:

$$\bar{L} = \sum_{i=0}^{k} iP_i = \sum_{i=0}^{k} i\frac{(1-\rho)}{(1-\rho^{k+1})}\rho^i, \rho \neq 1 \tag{4.1}$$

$$\bar{L} = \sum_{i=0}^{k} i\frac{1}{1+k}, \rho = 1 \tag{4.2}$$

Extending this concept to the parametric variations of the M/M/1/k queue, Figure 4.1 depicts the theoretical parametric Mean Queue Length, $\bar{L}$, with 20 discrete, evenly spaced variations of both k [1:20] and ρ [0.1:2.0].

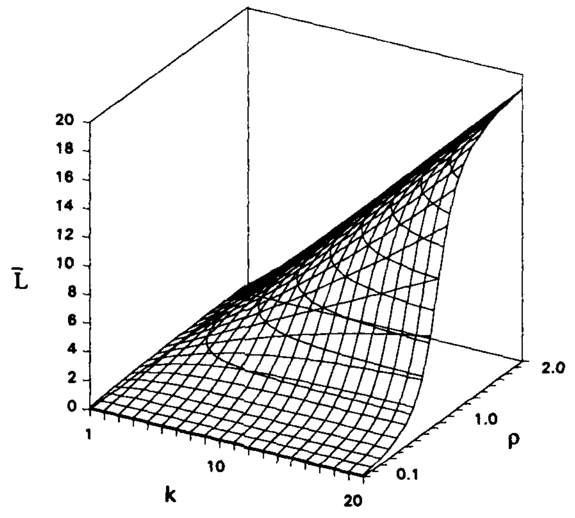


Figure 4.1 - Theoretical Parametric Mean Queue Length.

The theoretical shape of the variance surface can also be developed using a similar technique.

$$\sigma_{\bar{L}}^2 = \sum_{i=0}^{k}(i-\bar{L})^2 P_i = \sum_{i=0}^{k}(i-\sum_{j=0}^{k} j\frac{(1-\rho)}{(1-\rho^{k+1})}\rho^j)^2 \frac{(1-\rho)}{(1-\rho^{k+1})}\rho^i, \rho \neq 1 \tag{4.3}$$

$$\sigma_{\bar{L}}^2 = \sum_{i=0}^{k}(i-\bar{L})^2 P_i = \sum_{i=0}^{k}(i-\sum_{j=0}^{k} j\frac{1}{1+k})^2 \frac{1}{1+k}, \rho = 1 \tag{4.4}$$

Figure 4.2 illustrates the shape of the parametric variance surface for the M/M/1/k queue. The actual computed values of the variance are not illustrated. Direct value comparison between the theoretical and experimental variance is not practical. As a simulation with N replications proceeds, the variance of any performance measure should eventually tend to zero. The use of the following figure is that it provides a representation of the variance relationships within the parametric family. The Mean Queue Length parametric variance surface of either the event or time synchronous methods should take this shape during the course of the parallel simulation.

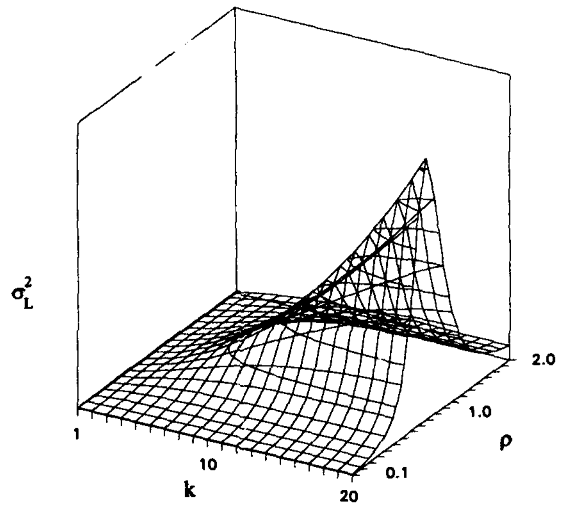


Figure 4.2 - Theoretical Parametric Variance Surface.

Figures 4.3 through 4.5 provide a graphical representation of the actual evolution of the Mean Queue Length and its corresponding variance for the event synchronous scheme. These *snapshots* were taken after 100, 1000 and 10,000 interevents. The time synchronous methods evolve in a similar manner. The event synchronous was chosen because it generates events at 100% efficiency and converged to the predicted shape at the fastest rate.

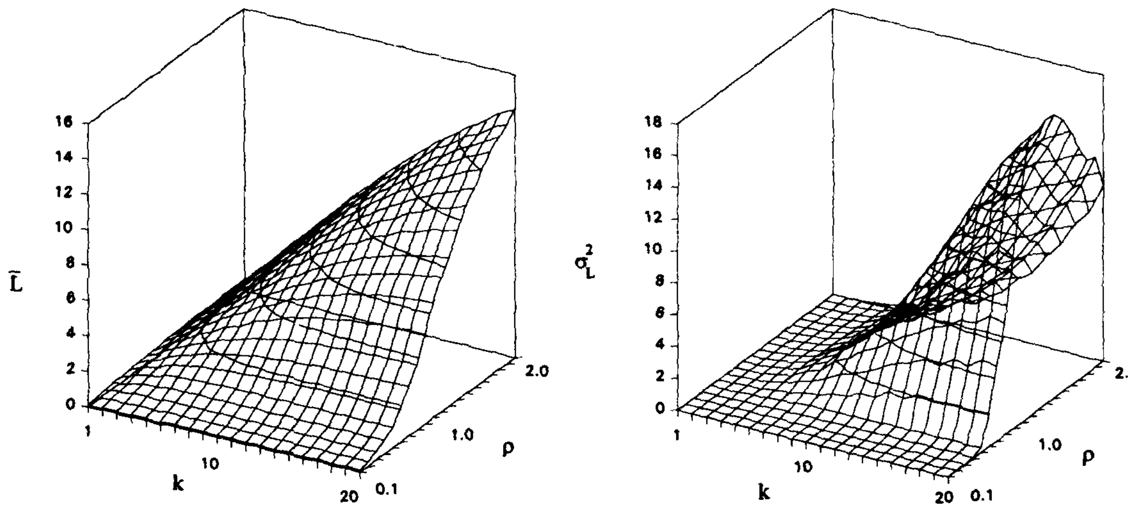


Figure 4.3 - Mean Queue Length and Variance After 100 Interevents.

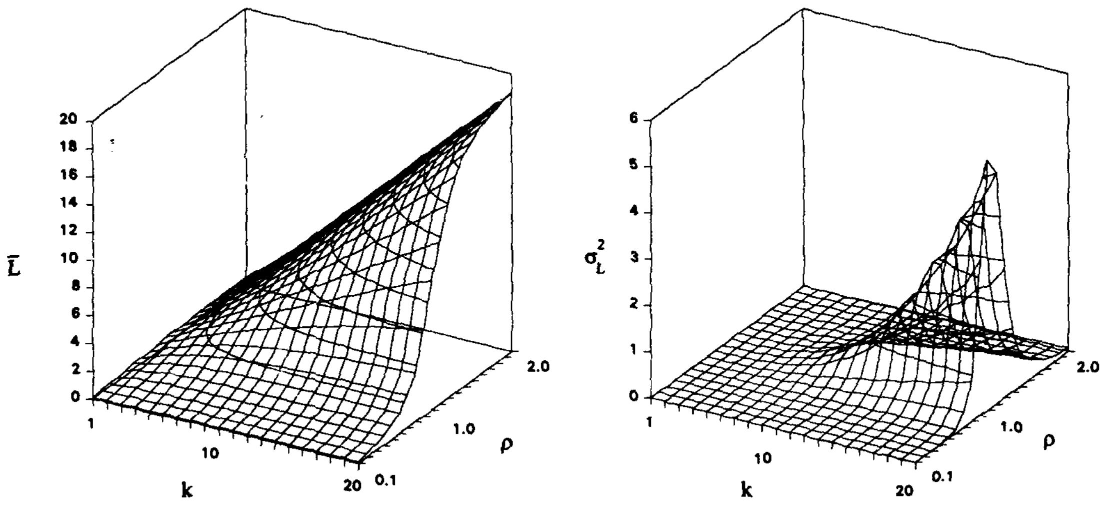
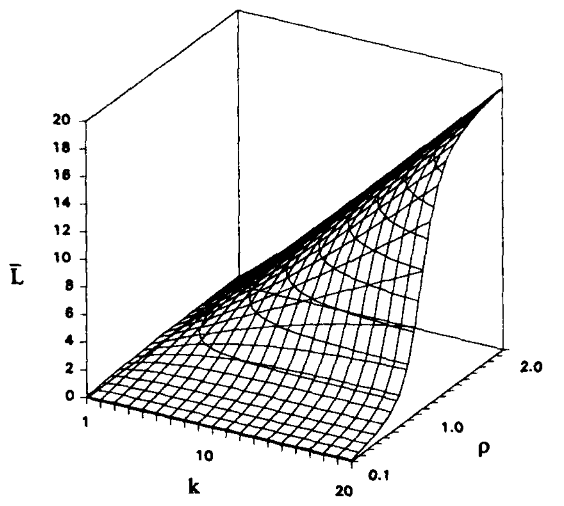


Figure 4.4 - Mean Queue Length and Variance After 1000 Interevents.



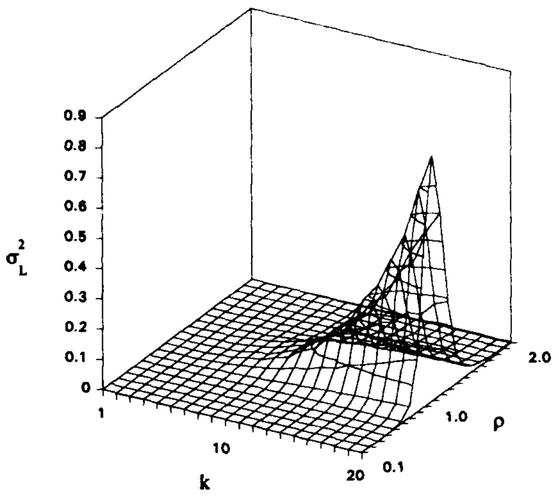


Figure 4.5 - Mean Queue Length and Variance After 10,000 Interevents.

It is obvious that the variance surface rapidly takes the proper shape. As the number of real events increase, the relative magnitude of the z-axis decreases as the individual system variances tend toward 0.

4.5.1 Variance Comparison.

The time and event synchronous parallel event generation algorithms are very similar. One of the major differences between the two is the fact that some events are thinned away or not used during a time synchronous simulation. In order to compare the two, the same number of total events (both real and null) should be generated. Even though some performance measures will quickly converge so as to preclude any meaningful comparisons, their respective variances may provide a clue as to their comparative performance. Equations 4.3 and 4.4 provide an analytical formulation of the queue length variance using the steady state probabilities of being in any state. The *Chapman-Kolmogorov* equations now provide a method for computing the n-step transition probabilities and hence the probability of being in any state after n steps or after n events are generated [Ross 1989].

$$P_{ij}^{n+m} = \sum_{k=0}^{\infty} P_{ik}^{n} P_{kj}^{m}, \text{ all } n, m \geq 0, \text{ all } i, j \tag{4.5}$$

Equation 4.5 represents the probability that starting in state i, the process will go to state j in n+m transitions through a path that takes it into state k at the nth transition. Essentially all possible intermediate paths are summed. All system simulations have the initial condition of time = 0 and state =0 after initialization. By conditioning on the initial state, the row vector $\mathbf{P_{0j}}$ of the computed transition matrix after the nth transition or event provides the necessary probabilities. This vector provides the probability of being in any particular state j, after n events. Ross demonstrates how the initial transition matrix may also be multiplied by itself n times to achieve these results. Equations 4.3 and 4.4 then become:

$$\sigma_L^2 = \sum_{i=0}^{k} (i - \bar{L})^2 P_i \Big|_{\text{n events generated}} = \sum_{i=0}^{k} (i - \sum_{j=0}^{k} jP_{0j}^{n})^2 P_{0i}^{n} \tag{4.6}$$

where P_{0j}^{n} represents the probability of being in state j after starting in state 0 and completing n transitions.

It makes intuitive sense that because no events are wasted using the event synchronous method, that its resulting performance measure variance will be lower than any of the time synchronous methods given the same number of events. Applying the n-step probability formulation to Equation 4.3 yields:

$$\left.\frac{\sigma^2_{L,\text{ (Event Synchronous)}}}{\sigma^2_{L,\text{ (Time Synchronous)}}}\right|_{\text{n events generated}} = \frac{\sum_{i=0}^{k} (i - \sum_{j=0}^{k} jP^{n}_{0j,\text{ Event}})^2 P^{n}_{0i,\text{ Event}}}{\sum_{i=0}^{k} (i - \sum_{j=0}^{k} jP^{n}_{0j,\text{ Time}})^2 P^{n}_{0i,\text{ Time}}} C \tag{4.7}$$

Since the two variances converge at a different rate, the ratio of their variances should converge to some constant, C. Kleinrock offers an approach that shows the n-step transition matrix converges exponentially to its limiting values at a rate of Determinant(**P**). All event and time synchronous methods have the same transition matrix determinant and thus converge at the same rate but as a function of the number of real events generated. Equation 4.7 then becomes:

$$\left.\frac{\sigma^2_{\text{L, (Event Synchronous)}}}{\sigma^2_{\text{L, (Time Synchronous)}}}\right|_{\text{n events generated}} = C \tag{4.8}$$

The event synchronous method produces events at a slightly different rate than any of the uniformized versions. Its effective event generation rate, $\hat{R}$, at processor i is:

$$\hat{R}_{\text{i, (Event Synchronous)}} = P_0\lambda_i + \sum_{j=1}^{k-1} P_j(\lambda_i + \mu_i) + P_k\mu_i \tag{4.9}$$

The time synchronous effective event generation rate is a constant because of the uniformization that occurs across all processors. If a front end method is chosen, then:

$$\hat{R}_{\text{(Time Synchronous)}} = \lambda_{\max} + \mu_{\max} \tag{4.10}$$

If the back end hybrid generation scheme is selected, then its rate is:

$$\hat{R}_{\text{(Time Synchronous)}} = \lambda_i + \mu_i \tag{4.11}$$

Consider any performance measure. Let $\overline{M}$ denote its mean. Integrating over equal time increments across multiple replications yields the following relationships:

$$\overline{M} = \frac{1}{nt}\int_0^{nt} m(t)dt = \frac{1}{nt}\left[\int_0^{t} m(t)dt + \int_t^{2t} m(t)dt + \ldots + \int_{(n-1)t}^{nt} m(t)dt\right]$$

$$\overline{M}_1 = \int_0^{t} m(t)dt$$

$$\overline{M} = \frac{1}{nt}\int_0^{nt} m(t)dt = \frac{1}{n}\left[\overline{M}_1 + .. + \overline{M}_n\right]$$

For large t, the $\overline{M}_i$ are iid. Similar relationships can now be developed for both the time and event synchronous variances.

$$\mathrm{Var}(M) = \frac{1}{nt}\int_0^{nt} (m(t)-\overline{M})^2 dt = \frac{1}{nt}\left[\int_0^{t} (m(t)-\overline{M})^2 dt + \int_t^{2t} (m(t)-\overline{M})^2 dt + \ldots + \int_{(n-1)t}^{nt} (m(t)-\overline{M})^2 dt\right]$$

$$\mathrm{Var}(M_{Event}) = \frac{\hat{R}_{Event}}{n}\int_0^{\frac{n}{\hat{R}_{Event}}} (m(t)-\overline{M})^2 dt = \frac{1}{nt}\left[\int_0^{\frac{1}{\hat{R}_{Event}}} (m(t)-\overline{M})^2 dt + \int_{\frac{1}{\hat{R}_{Event}}}^{\frac{2}{\hat{R}_{Event}}} (m(t)-\overline{M})^2 dt + \ldots + \int_{\frac{n-1}{\hat{R}_{Event}}}^{\frac{n}{\hat{R}_{Event}}} (m(t)-\overline{M})^2 dt\right]$$

$$\mathrm{Var}(M_{Time}) = \frac{\hat{R}_{Time}}{n}\int_0^{\frac{n}{\hat{R}_{Time}}} (m(t)-\overline{M})^2 dt = \frac{1}{nt}\left[\int_0^{\frac{1}{\hat{R}_{Time}}} (m(t)-\overline{M})^2 dt + \int_{\frac{1}{\hat{R}_{Time}}}^{\frac{2}{\hat{R}_{Time}}} (m(t)-\overline{M})^2 dt + \ldots + \int_{\frac{n-1}{\hat{R}_{Time}}}^{\frac{n}{\hat{R}_{Time}}} (m(t)-\overline{M})^2 dt\right]$$

Assuming for large time intervals, the variance segments are once again iid, then the ratio of the variances (Equation 4.8) reduces to:

$$\frac{\mathrm{Var}(M_{\mathrm{Event}})}{\mathrm{Var}(M_{\mathrm{Time}})} = \frac{\hat{R}_{\mathrm{Event}}}{\hat{R}_{\mathrm{Time}}} = \frac{P_0\lambda_i + \sum_{j=1}^{k-1} P_j(\lambda_i + \mu_i) + P_k\mu_i}{R} = C \tag{4.12}$$

For each of the time synchronous generation methods, after some lengthy algebraic manipulation, C equals the actual processor efficiency as computed in Chapter 3.0 independent of the performance measure being measured. Therefore, given an event synchronous variance, the resulting time synchronous variance after an equivalent number of generated events (both real and null) can be predicted.

4.5.2 Simulation Results.

Section 4.5.1 showed that the predicted efficiency also could determine the relative variance between any time synchronous method to the event synchronous method for any performance measure. 30 serial replications were analyzed for the front end time synchronous method and 30 simultaneous replications for all others. Table 4.1 provides a comparison of the predicted to actual average efficiency for these simulations.

	Predicted $\bar{\varepsilon}$	1000 Events	10,000 Events	50,000 Events
Back End Hybrid	0.687304	0.690363	0.687876	0.687321
Front End Hybrid	0.438638	0.439378	0.436140	0.436341
μ Constant	0.488126	0.489237	0.488354	0.487979
λ Constant	0.144324	0.145482	0.144146	0.144040

Table 4.1 - Predicted and Measured Efficiency.

Table 4.1 demonstrates that the average processor efficiency quickly converges to its predicted value.

Figure 4.6 provides a graphical representation of the efficiency surface of the back end hybrid method. The viewpoint for this surface is oriented such that the overall trend can be observed with each line representing a discrete buffer size, k.

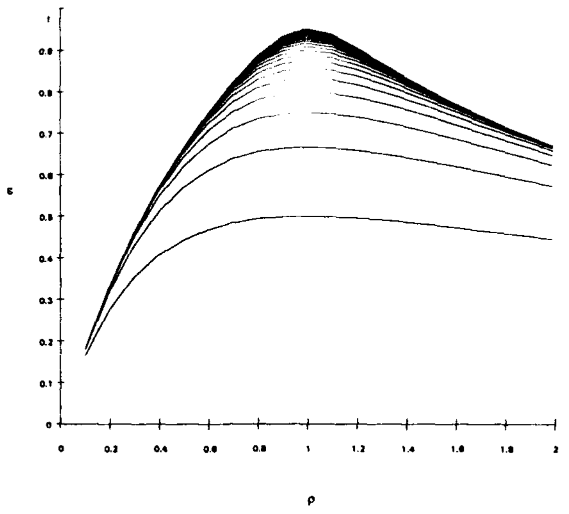


Figure 4.6 - Predicted Efficiency - Back End Hybrid.

The next step is to compute the variance ratio and compare it to the actual efficiency. This should confirm the analytical conclusions as to whether or not the predicted efficiency can be used to predict the relative variance difference between the methods. The following two sets of figures are actual measured variance ratios between the back end hybrid and event synchronous methods for 30 and 150 replications respectively. Comparing these figures with Figure 4.6 shows that as the number of events and/or replications increases, the variance ratio surface tends towards its predicted form.

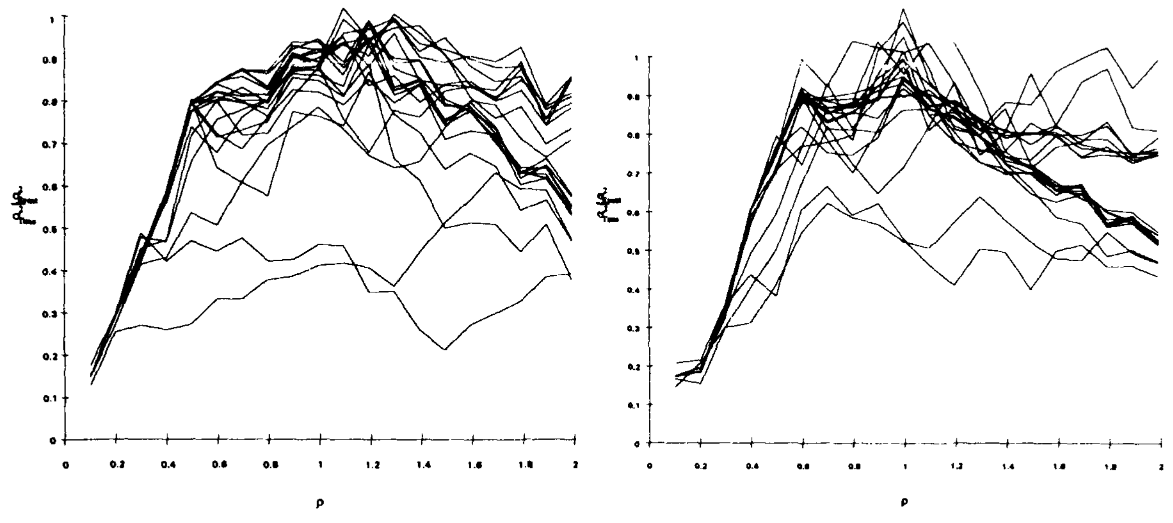


Figure 4.7 - Back End Hybrid Variance Ratio; 30 Replications, 1000 and 10,000 Events.

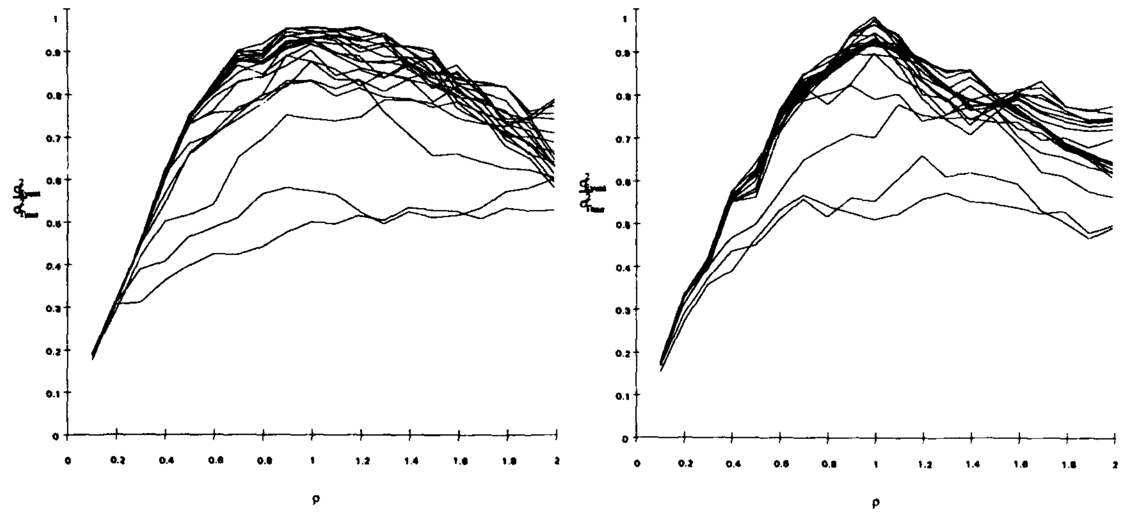


Figure 4.8 - Back End Hybrid Variance Ratio; 150 Replications, 1000 and 10,000 Events.

The front end hybrid predicted efficiency is given in Figure 4.9. This method has roughly the same shape as the back end but with noticeably lower overall efficiency.

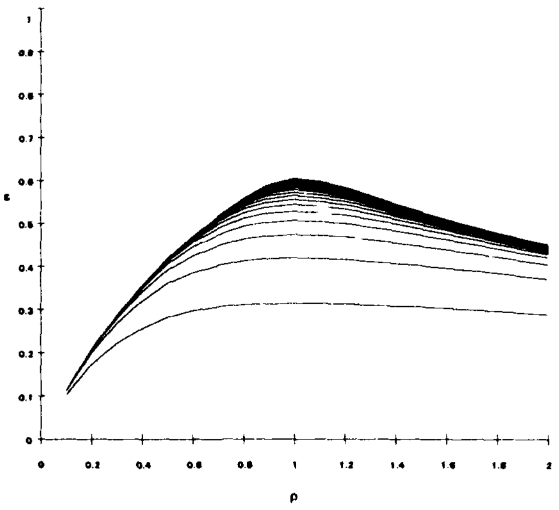


Figure 4.9 - Predicted Efficiency - Front End Hybrid.

The following figures depict the variance ratio of the front end hybrid to the event synchronous and its evolution as both the number of events and/or replications are increased. Once again, compare the following figures with Figure 4.9 to see formulation of the predicted shape.

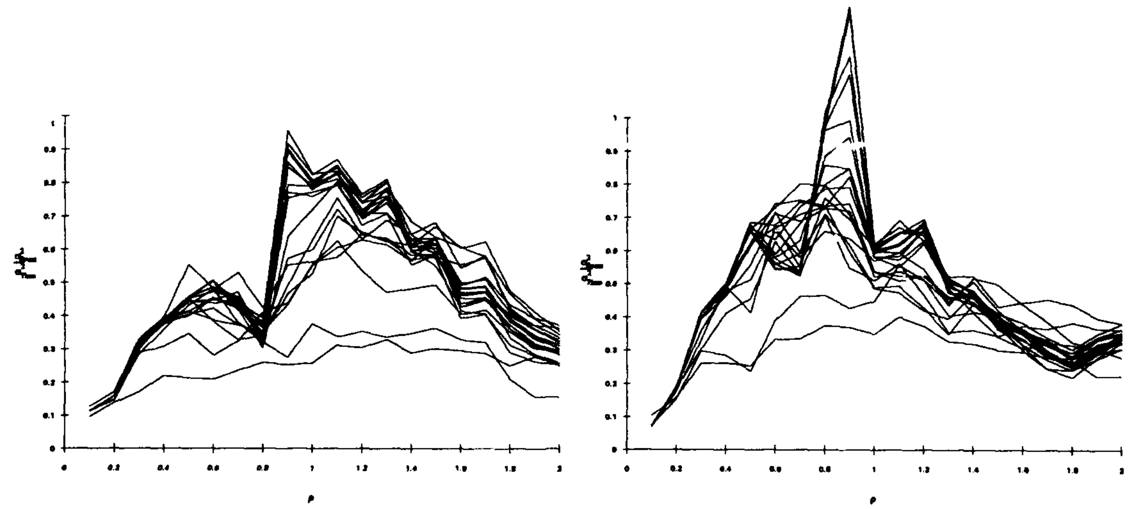


Figure 4.10 - Front End Hybrid Variance Ratio; 30 Replications, 1000 and 10,000 Events.

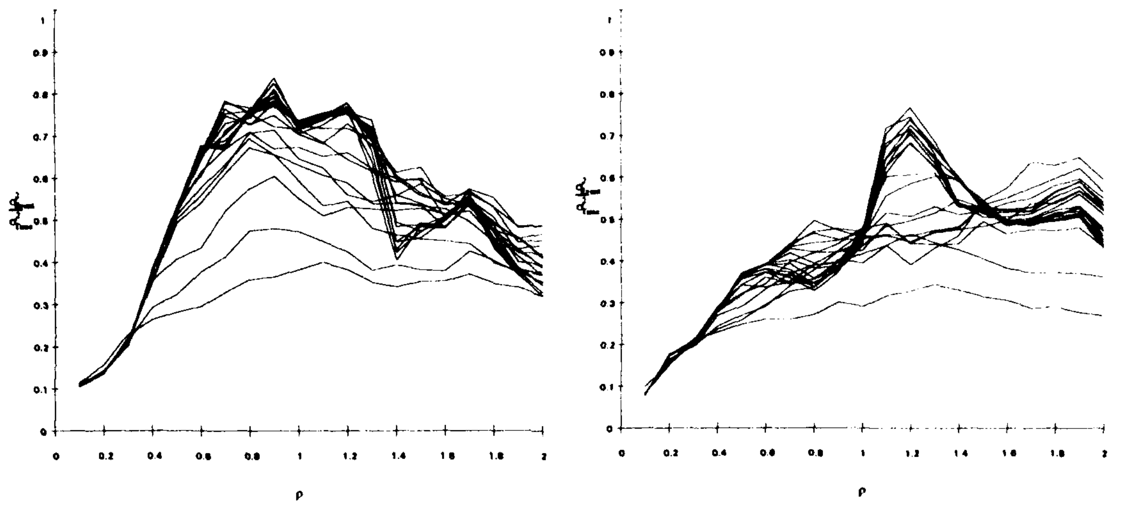


Figure 4.11 - Front End Hybrid Variance Ratio; 150 Replications, 1000 and 10,000 Events.

The next comparison involves the μ constant parameterization method. Figure 4.12 provides the theoretical efficiency for the same range as used in the previous two cases. Figures 4.13 and 4.14 depict the service rate constant scheme variance ratio as it evolves with regards to number of events and efficiency. Note that this method is much more sensitive to the number of events generated as opposed to increasing replications.

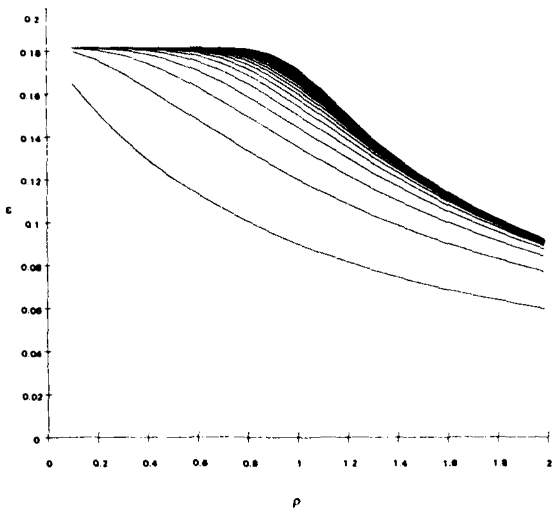


Figure 4.12 - Predicted Efficiency - Service Rate Constant.

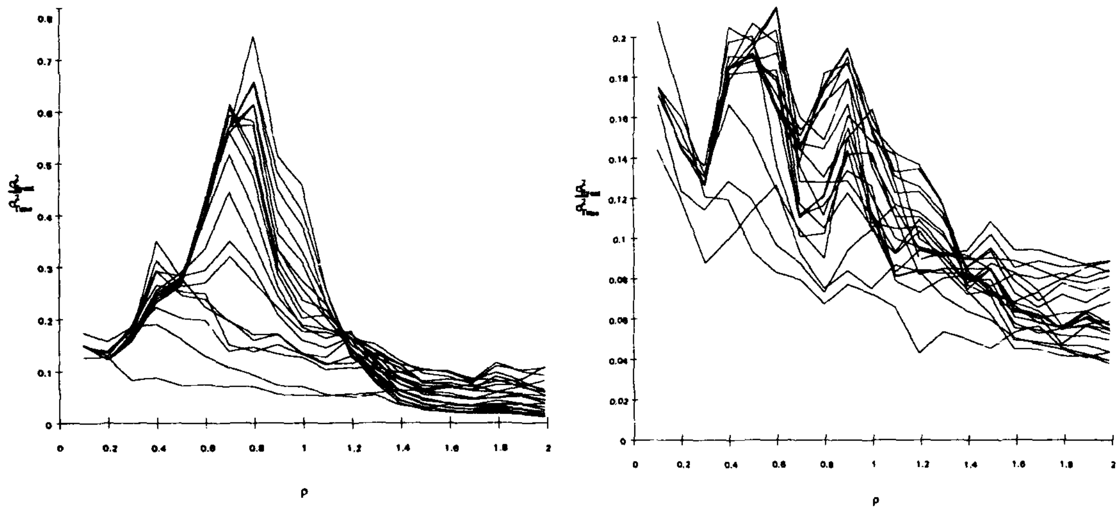


Figure 4.13 - Service Rate Constant Variance Ratio; 30 Replications, 1000 and 10,000 Events.

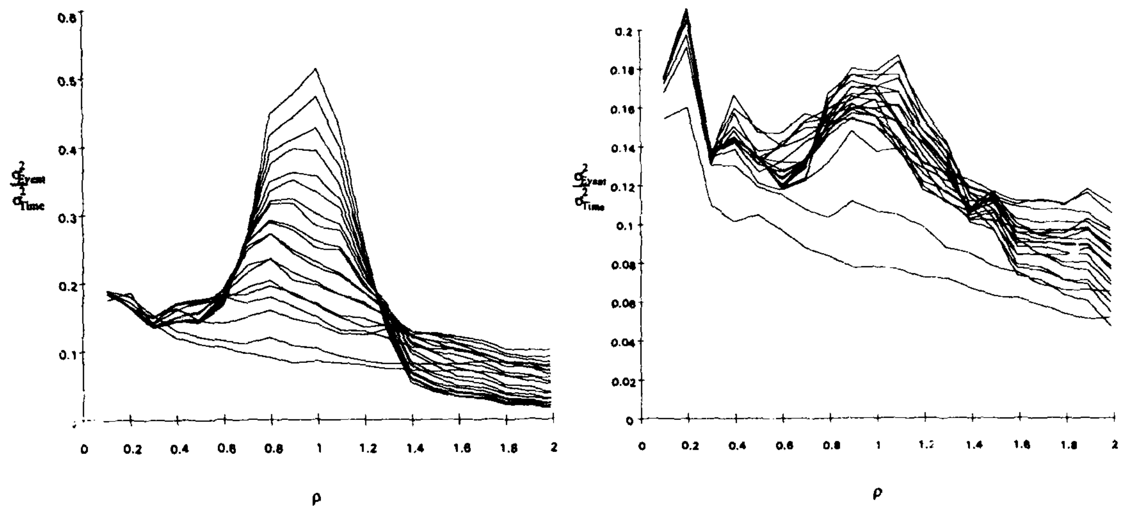


Figure 4.14 - Service Rate Constant Variance Ratio; 150 Replications, 1000 and 10,000 Events.

The last case to be considered is the λ constant, which has the lowest average efficiency.

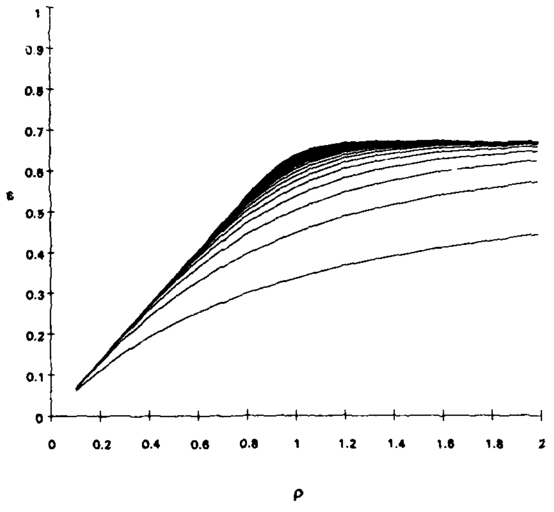


Figure 4.15 - Predicted Efficiency - Arrival Rate Constant.

Figures 4.16 and 4.17 provide the variance ratios for this method.

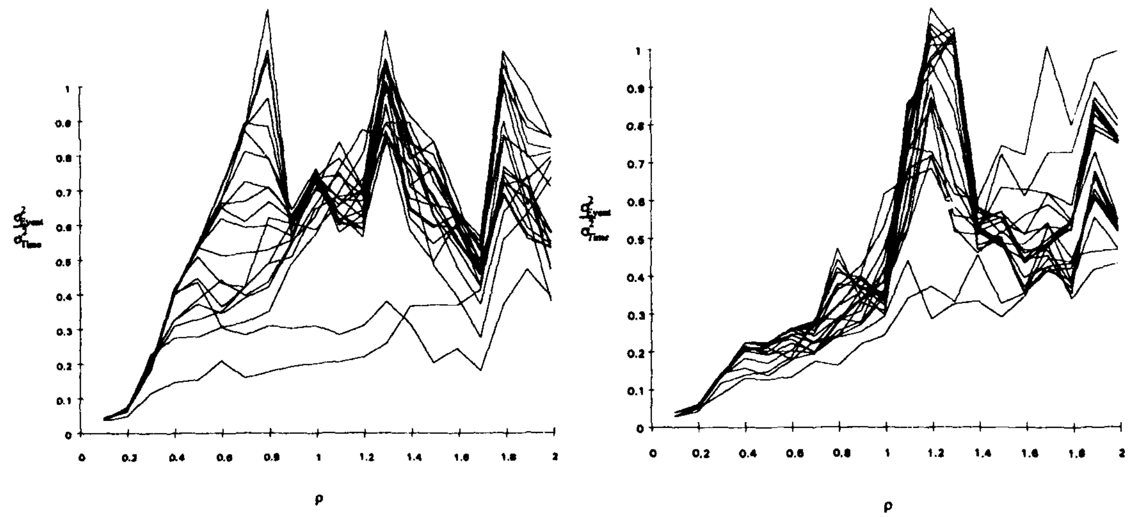


Figure 4.16 - Arrival Rate Constant Variance Ratio; 30 Replications, 1000 and 10,000 Events.

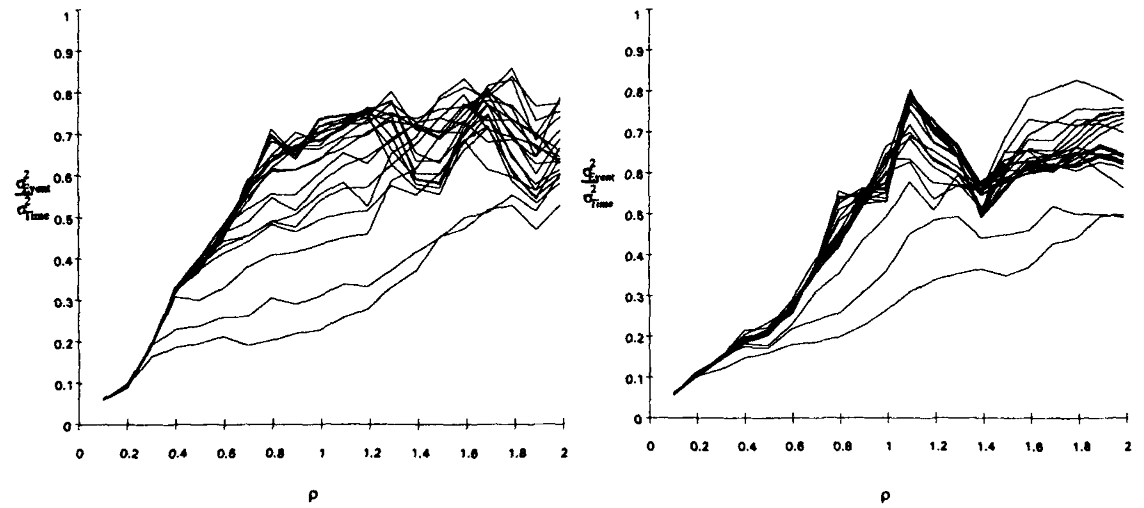


Figure 4.17 - Arrival Rate Constant Variance Ratio; 150 Replications, 1000 and 10,000 Events.

The following table depicts the average variance ratio for the different time synchronous parameterization methods.

		Variance Ratio			
	Predicted $\bar{\varepsilon}$	30 Reps, 1K Events	30 Reps, 10K Events	150 Reps, 1K Events	150 Reps, 10K Events
Back End Hybrid	0.6873	0.6754	0.6779	0.7023	0.6788
Front End Hybrid	0.4386	0.4536	0.4539	0.4963	0.4137
μ Constant	0.4881	0.5437	0.4161	0.5156	0.4557
λ Constant	0.1443	0.1694	0.1135	0.1531	0.1278

Table 4.2 - Predicted Efficiency and Measured Variance Ratio.

One can conclude from this table as well as the previous figures of the variance ratios, that processor efficiency can be used as a means to estimate the relative variance of any time synchronous method as compared to the event synchronous with some stochastic variation. For complicated systems, inefficient events can be directly counted as a simulation progresses to form the same estimates.

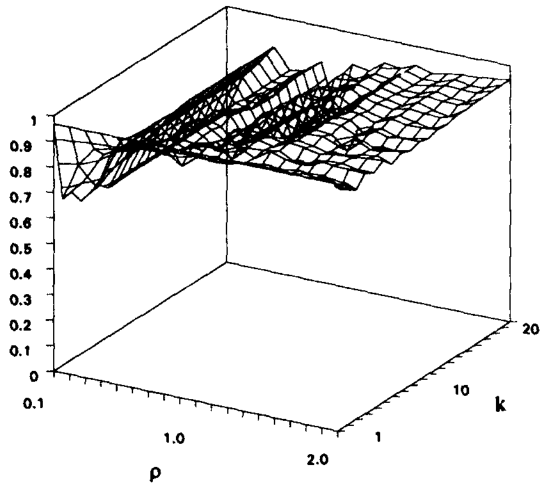
4.5.3 Correlation and Variance Reduction.

Common Random Numbers is a means of stochastically coupling parametric variations and is one of a variety of variance reduction methods that seem ideally suited for the SIMD machine. By using the same stream of random variables, positive correlation between members of a parametric family can be induced. The resulting variance of the difference between members of a parametric family would then be lower than the individual variances of each. This variance reduction technique is well suited in developing the ordinal optimality for a parametric family of systems. Either correlation or covariance could be measured directly to ascertain the effectiveness of any stochastic coupling. Correlation is a dimensionless quantity that is normalized to fall in the range [-1,1]. The possible values of covariance are directly related to the values of the actual variances from which they are computed. Correlation is measured during the parallel simulation through the use of the following equations:

$$\sigma^2_{(L_i - L_{i-1})} = \sigma^2_{L_i} + \sigma^2_{L_{i-1}} - 2\sigma_{(L_i, L_{i-1})}, i = \rho_2 .. \rho_{20} \tag{4.13}$$

$$\mathbf{Correlation}(L_i, L_{i-1}) = \frac{\sigma_{(L_i, L_{i-1})}}{\sigma_{L_i}\sigma_{L_{i-1}}} \tag{4.14}$$

Covariance could be quickly computed if the need arises to use its value. The following table provides a listing of the realized average correlation at the same event snapshots as Table 4.1. Once again these measurements were taken over 30 replications. As the number of replications was increased above 30, there was no noted significant difference in the amount of induced correlation. The data in the following figures and tables represent the individual and average correlation achieved over successive differences of ρ and k respectively. Each plot contains 380 data points (19 ρ, 20 k or 20 ρ, 19 k). The tables also provide a measure of the variance of the average achieved correlation for each of these methods. Figure 4.18 and Table 4.3 detail the correlation for the event synchronous. The plot on the left is for the difference of successive ρ's, 0.1 apart, and the one on the right for successive k's. Note that the successive ρ correlation is higher than the successive k.



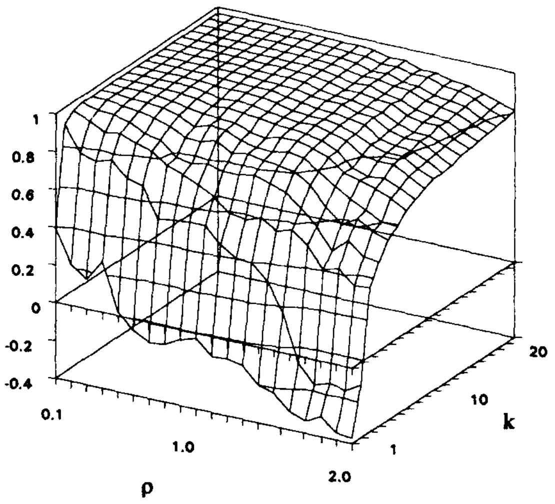


Figure 4.18 - Event Synchronous - Achieved Correlation.

	$E\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$E\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$
1,000 Events	0.8675	0.0079	0.7965	0.0763
10,000 Events	0.8802	0.0034	0.7879	0.0740
50,000 Events	0.8553	0.0087	0.7396	0.0972

Table 4.3 - Event Synchronous - Average Achieved Correlation.

The following two figures and tables contain the correlation achieved for the two hybrid methods. Higher correlation is achieved using the front end method for successive ρ's and is roughly the same for successive k's. Once again the plot on the left is for the successive ρ's.

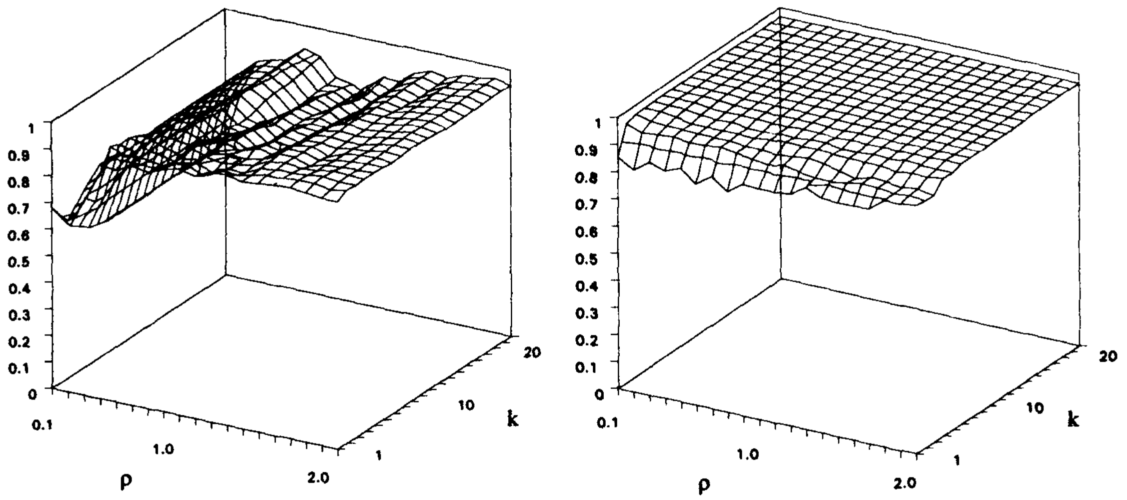


Figure 4.19 - Front End Hybrid - Achieved Correlation.

	$E\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$E\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$
1,000 Events	0.8956	0.0034	0.9540	0.0006
10,000 Events	0.8864	0.0050	0.9550	0.0006
50,000 Events	0.8952	0.0047	0.9486	0.0016

Table 4.4 - Front End Hybrid - Average Achieved Correlation.

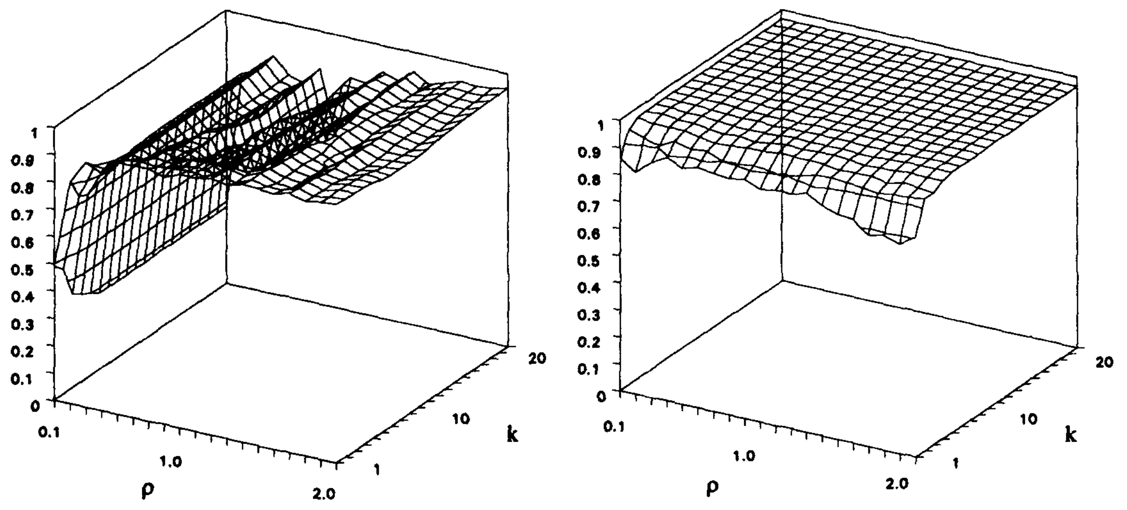


Figure 4.20 - Back End Hybrid - Achieved Correlation.

	$E\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$E\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$
1,000 Events	0.8491	0.0203	0.9539	0.0008
10,000 Events	0.8632	0.0066	0.9486	0.0017
50,000 Events	0.8291	0.0166	0.9479	0.0017

Table 4.5 - Back End Hybrid - Average Achieved Correlation.

The next two figures and tables provide the correlation for the μ constant methods. Once again higher correlation is achieved using the front end technique for successive ρ's, represented by the plot on the left of Figures 4.21 and 4.22.

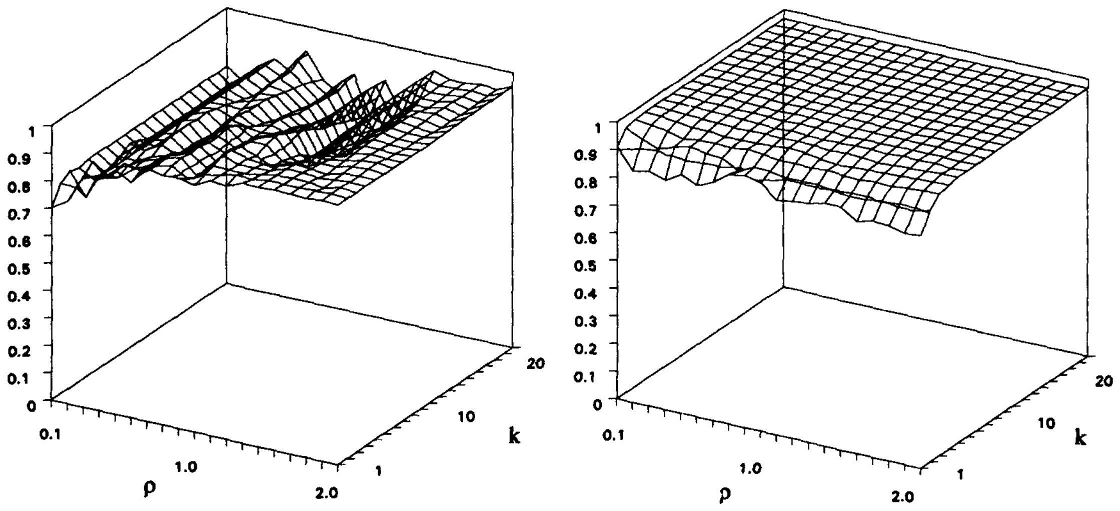


Figure 4.21 - Service Rate Constant - Front End Marked - Achieved Correlation.

	$E\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$E\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$
1,000 Events	0.8883	0.0075	0.9536	0.0007
10,000 Events	0.8942	0.0020	0.9558	0.0007
50,000 Events	0.8823	0.0036	0.9501	0.0020

Table 4.6 - Service Rate Constant - Front End Marked - Average Achieved Correlation.

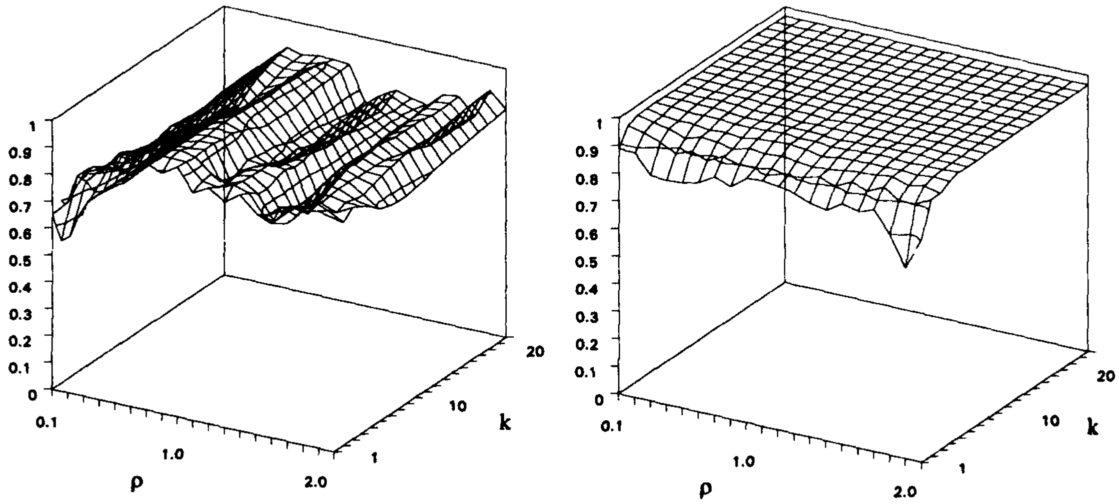


Figure 4.22 - Service Rate Constant - Back End Marked - Achieved Correlation.

	$E\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$E\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$
1,000 Events	0.8202	0.0067	0.9526	0.0010
10,000 Events	0.8104	0.0032	0.9558	0.0012
50,000 Events	0.8040	0.0068	0.9501	0.0016

Table 4.7 - Service Rate Constant - Back End Marked - Average Achieved Correlation.

The following two figures and tables provide the correlation achieved for the λ constant schemes.

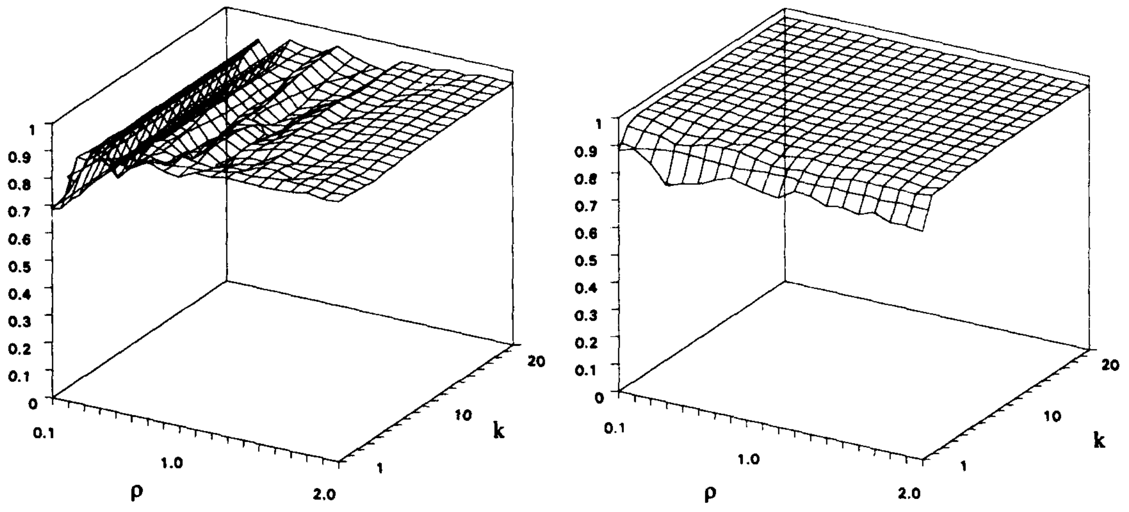


Figure 4.23 - Arrival Rate Constant - Front End Marked - Achieved Correlation.

	$E\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$E\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$
1,000 Events	0.8835	0.0084	0.9552	0.0007
10,000 Events	0.8784	0.0052	0.9510	0.0015
50,000 Events	0.8911	0.0057	0.9511	0.0012

Table 4.8 - Arrival Rate Constant - Front End Marked - Average Achieved Correlation.

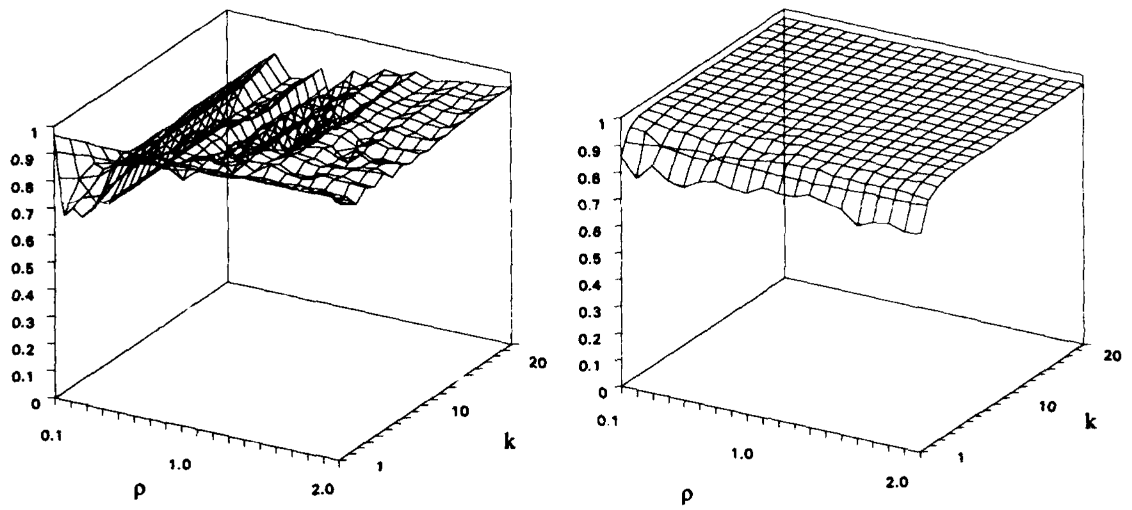


Figure 4.24 - Arrival Rate Constant - Back End Marked - Achieved Correlation.

	$E\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{\rho_i,\rho_{i+1}}}{\sigma_{\rho_i}\sigma_{\rho_{i+1}}}\right)$	$E\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$	$\sigma^2\left(\frac{\sigma_{k_i,k_{i+1}}}{\sigma_{k_i}\sigma_{k_{i+1}}}\right)$
1,000 Events	0.8793	0.0039	0.9528	0.0010
10,000 Events	0.8845	0.0052	0.9497	0.0017
50,000 Events	0.8812	0.0035	0.9469	0.0029

Table 4.9 - Arrival Rate Constant - Back End Marked - Average Achieved Correlation.

The principle reason for inducing this high correlation is to significantly reduce the variance of successive differences. All of the methods are able to produce relatively high correlation. Because the front end methods produce marked events to the individual processors, the amount of correlation using this technique is highest for successive ρ's. There is no marked difference for successive k's. Taking successive differences of buffer sizes provides higher correlation which is roughly the same across all methods for this particular performance measure (queue length). Overall, achieved correlation is a function of the performance measure and parametric variation over which differences are to be taken..

The following tables depict the variance improvement achieved with this high induced correlation. The first portion of each table represents the average variance over the 400 parametric variations performed for each of the methods after 30 replications of each. Because of the high induced correlation, there is significant variance improvement for all of the methods.

The first table presented contains the experimental results for the event synchronous method. The first two columns represent the average queue length variance and the variance of the queue length variances. 400 parametric variations were used to generate these values. Recall Figure 4.2. These values would represent the average of every point represented in that figure. The next set of columns give the variance mean and variance of the variances of the successive differences along the ρ or k axes. 380 points were used to generate this data. Because of the event synchronous 100% efficiency, both the variance and successive variance differences averages are lower than any other method.

	$E[\sigma_L^2]$	$\sigma^2[\sigma_L^2]$
1,000 Events	0.3501	0.7561
10,000 Events	0.0381	0.0101
50,000 Events	0.0059	0.0002

	$E[\sigma_{L,\Delta\rho}^2]$	$\sigma^2[\sigma_{L,\Delta\rho}^2]$	$E[\sigma_{L,\Delta k}^2]$	$\sigma^2[\sigma_{L,\Delta k}^2]$
1,000 Events	0.1019	0.0728	0.0214	0.0008
10,000 Events	0.0115	0.0009	0.0017	5.50 E-06
50,000 Events	0.0019	2.67 E-05	0.0004	2.71 E-07

Table 4.10 - Event Synchronous - Variance Comparison.

The next two tables give the experimental results for the two hybrid methods. Event though the front end hybrid was able to achieve successive ρ higher correlation, it still did not perform as well as the back end hybrid.

	$E[\sigma^2_L]$	$\sigma^2[\sigma^2_L]$
1,000 Events	0.5044	1.1643
10,000 Events	0.0590	0.0202
50,000 Events	0.0119	0.0009

	$E[\sigma^2_{L,\Delta\rho}]$	$\sigma^2[\sigma^2_{L,\Delta\rho}]$	$E[\sigma^2_{L,\Delta k}]$	$\sigma^2[\sigma^2_{L,\Delta k}]$
1,000 Events	0.1178	0.0884	0.0070	0.0002
10,000 Events	0.0161	0.0023	0.0007	2.51 E-06
50,000 Events	0.0032	8.55 E-05	0.0001	6.46 E-08

Table 4.11 - Front End Hybrid - Variance Comparison.

	$E[\sigma^2_L]$	$\sigma^2[\sigma^2_L]$
1,000 Events	0.3890	0.8565
10,000 Events	0.0416	0.0115
50,000 Events	0.0066	0.0002

	$E[\sigma^2_{L,\Delta\rho}]$	$\sigma^2[\sigma^2_{L,\Delta\rho}]$	$E[\sigma^2_{L,\Delta k}]$	$\sigma^2[\sigma^2_{L,\Delta k}]$
1,000 Events	0.1164	0.0924	0.0049	8.43 E-05
10,000 Events	0.0133	0.0012	0.0004	6.68 E-07
50,000 Events	0.0022	3.67 E-05	7.97 E-05	2.32 E-08

Table 4.12 - Back End Hybrid - Variance Comparison.

The next two tables show the results for the μ constant scheme. Note that even though the efficiency of this method is the same for either front or back end marking technique, the front end was able to outperform the back end because of the higher induced correlation. Also, note that the front end hybrid was able to outperform the back end μ constant in the variance of the differences even though it has lower overall efficiency. The achieved correlation for the front end hybrid was able to overcome its lower efficiency to produce lower variance of successive differences estimates.

	$E[\sigma^2_L]$	$\sigma^2[\sigma^2_L]$
1,000 Events	0.5287	1.7570
10,000 Events	0.0401	0.0085
50,000 Events	0.0102	0.0007

	$E[\sigma^2_{L,\Delta\rho}]$	$\sigma^2[\sigma^2_{L,\Delta\rho}]$	$E[\sigma^2_{L,\Delta k}]$	$\sigma^2[\sigma^2_{L,\Delta k}]$
1,000 Events	0.1348	0.1372	0.0068	0.0001
10,000 Events	0.0121	0.0010	0.0009	3.80 E-06
50,000 Events	0.0034	9.93 E-05	0.0001	4.97 E-08

Table 4.13 - Service Rate Constant - Front End Marked - Variance Comparison.

	$E[\sigma^2_L]$	$\sigma^2[\sigma^2_L]$
1,000 Events	0.4737	1.0435
10,000 Events	0.0433	0.0112
50,000 Events	0.0078	0.0003

	$E[\sigma^2_{L,\Delta\rho}]$	$\sigma^2[\sigma^2_{L,\Delta\rho}]$	$E[\sigma^2_{L,\Delta k}]$	$\sigma^2[\sigma^2_{L,\Delta k}]$
1,000 Events	0.1610	0.1223	0.0068	0.0002
10,000 Events	0.0182	0.0018	0.0007	2.24 E-06
50,000 Events	0.0037	8.41 E-05	0.0002	1.92 E-07

Table 4.14 - Service Rate Constant - Back End Marked - Variance Comparison.

The next set of tables provide similar data for the λ constant. The efficiency for this method is so low in comparison to the other methods, that no degree of correlation would assist it in overtaking any of the other methods in variance performance.

	$E[\sigma_L^2]$	$\sigma^2[\sigma_L^2]$
1,000 Events	1.1011	4.1955
10,000 Events	0.2056	0.1822
50,000 Events	0.0394	0.0083

	$E[\sigma_{L,\Delta\rho}^2]$	$\sigma^2[\sigma_{L,\Delta\rho}^2]$	$E[\sigma_{L,\Delta k}^2]$	$\sigma^2[\sigma_{L,\Delta k}^2]$
1,000 Events	0.2196	0.1984	0.0235	0.0006
10,000 Events	0.0500	0.0178	0.0029	3.33 E-05
50,000 Events	0.0098	0.0007	0.0005	9.99 E-07

Table 4.15 - Arrival Rate Constant - Front End Marked - Variance Comparison.

	$E[\sigma_L^2]$	$\sigma^2[\sigma_L^2]$
1,000 Events	1.4614	5.0765
10,000 Events	0.1972	0.1987
50,000 Events	0.0376	0.0078

	$E[\sigma_{L,\Delta\rho}^2]$	$\sigma^2[\sigma_{L,\Delta\rho}^2]$	$E[\sigma_{L,\Delta k}^2]$	$\sigma^2[\sigma_{L,\Delta k}^2]$
1,000 Events	0.1800	0.0794	0.0250	0.0007
10,000 Events	0.0494	0.0172	0.0025	1.80 E-05
50,000 Events	0.0114	0.0011	0.0005	8.38 E-07

Table 4.16 - Arrival Rate Constant - Back End Marked - Variance Comparison.

The best method is the event synchronous followed closely by the back end hybrid. Higher correlation is achieved using the front end marking methods as compared to the back end methods. Some of the efficiency loss is then overcome and inefficient methods may achieve better successive variance difference performance over more efficient methods on the back end. It is obvious that the SIMD architecture can be exploited to induce high correlation. Comparison of different systems can then be accomplished in even shorter computational time using the method of successive differences as opposed to brute force computation of the actual performance measure.

Another measure available to compare the relative performance of the differing methods is comparison of the relevant *Coefficients of Variation*. This measure is useful in that it scales each variance by the inverse of its respective performance measure mean at that point in the simulation. The following table represent once again the average and variance of the 400 data values for the 20 by 20 parametric family of systems.

	E(CV(L))	σ^2(CV(L))
1,000 Events	0.0912	0.0045
10,000 Events	0.0297	0.0006
50,000 Events	0.0118	8.0 E-05

Table 4.17 - Event Synchronous - Coefficient of Variation.

The next two tables compare the hybrid methods. Back end marking results in the overall lower variability.

	E(CV(L))	σ^2(CV(L))
1,000 Events	0.1449	0.0102
10,000 Events	0.0443	0.0009
50,000 Events	0.0201	0.0002

Table 4.18 - Front End Hybrid - Coefficient of Variation.

	E(CV(L))	σ^2(CV(L))
1,000 Events	0.1120	0.0054
10,000 Events	0.0362	0.0007
50,000 Events	0.0144	9.6 E-05

Table 4.19 - Back End Hybrid - Coefficient of Variation.

The following two table compare the service rate constant methods. There is not as clear a distinction between the two methods.

	E(CV(L))	σ^2(CV(L))
1,000 Events	0.1484	0.0117
10,000 Events	0.0451	0.0012
50,000 Events	0.0202	0.0005

Table 4.20 - Service Rate Constant - Front End Marked - Coefficient of Variation.

	E(CV(L))	σ^2(CV(L))
1,000 Events	0.1376	0.0100
10,000 Events	0.0429	0.0011
50,000 Events	0.0192	0.0002

Table 4.21 -Service Rate Constant - Back End Marked - Coefficient of Variation.

The next two tables compare the arrival rate constant. Note now that front end marking results in lower variability.

	E(CV(L))	σ^2(CV(L))
1,000 Events	0.1891	0.0136
10,000 Events	0.0659	0.0013
50,000 Events	0.0304	0.0004

Table 4.22 - Arrival Rate Constant - Front End Marked - Coefficient of Variation.

	E(CV(L))	σ^2(CV(L))
1,000 Events	0.2179	0.0154
10,000 Events	0.0728	0.0028
50,000 Events	0.0310	0.0005

Table 4.23 - Arrival Rate Constant - Back End Marked - Coefficient of Variation.

As to be expected, the Coefficient of Variation is a direct function of the number of events and efficiency. For methods with high efficiency, back end marking tends to have lower variability. It is apparent however, that lower efficiency methods are less variable using front end methods. The minor differences between the λ and μ constant on the front end and back end are due to random number generation starting seed differences.

4.8 Speedup.

The justification for parallel computing is the relative comparative performance improvement over conventional serial computing techniques. Comparison of the timing and efficiency of both the time (both front and back end) and event synchronous methods on the CM-2 is crucial in determining which algorithm provides the best overall performance. Recall that parallel simulation speedup on the SIMD architecture is governed by the following speedup equation as developed in Section 1.5.

The event and time synchronous algorithms are essentially identical during the INITIALIZE and WRAP-UP portions. The crucial differences occur during the event generation phase of the simulation which is the bulk of the computational effort. There are several major algorithmic differences. The first is in the computation of the interevent time. The time synchronous methods use one of the following equations to compute this interval depending on whether the process is marked on the front end or at the local processors.

$$\Delta t = -\frac{1}{\lambda_{max} + \mu_{max}} \ln(1-U) \tag{4.15}$$

$$\Delta t = -\frac{1}{\lambda_i + \mu_i} \ln(1-U) \tag{4.16}$$

The event synchronous method is slightly different and requires two extra instructions in parallel to determine the next event type. Computation of the local interevent time is also a function of the current state of the system.

$$\Delta t_i = -\frac{1}{\lambda_i I_{k,i} + \mu_i I_{0,i}} \ln(1-U) \tag{4.17}$$

$I_0 = 0$ if the system is in state 0, 1 otherwise, and $I_k = 0$ if the system is in state k, 0 otherwise. On the CM-2 bit-serial processor, this computation is comparatively long as compared to Equation 4.16. Computational effort is decreased by the use of the CM *where* command which is a masking command at the local processor level as compared to local processor floating point multiplies and divides as indicated in Equation 4.17. The *where* command directs a conditional array assignment across all processors wherever the statement within the parentheses is evaluated as true. For the following algorithm, assume that all elements identified in **bold** are parallel arrays with the same shape and layout and already received their initial values during INITIALIZE.

	Event Synchronous	Time Synchronous

LOOP:	**rate=lambda+mu**
	rate1=rate
	where(state=0)rate1=lambda
	where(state=k)rate1=mu
	d_time=-1/rate1*ln(1-U)
	Perform other calculations
	Go to LOOP
		rate=lambda_max+mu_max
		
		d_time =-1/rate*ln(1-U)
		Perform other calculations
		Go to LOOP

Algorithmic differences must be accounted for in order to perform an accurate comparison of the two methods. Recall that for the time synchronous methods, there is an average efficiency, $\bar{\varepsilon}$, associated with each which is a function of the method selected (λ constant, μ constant, or hybrid), the location of event marking (front or back end) and as the range of parametric systems to be considered.

Another major difference to consider is whether or not to perform multiple replications. Multiple replications require the generation in parallel of the random numbers required during each event generation and should generally be performed using back end marking techniques. At each processor, the <u>two</u> required random numbers could be generated in parallel. Serial replications allow the generation of the <u>three</u> required random numbers on the front end which are then subsequently broadcast to all processors performing the simulations.

4.6.1 Simultaneous Replications Comparison - Back End Methods.

The following table provides experimental timing for a sequence of parallel simulations in which unmarked events were broadcast from the front end. Each simulation was performed on a 10 by 20 parametric family of M/M/1/k queues where buffer size, k, was varied along the one axis and ρ along a second. The number of stochastic variants was varied along a third axis to provide these timing results. This data was measured after the generation of 1000 interevents for each algorithm.

Multiplying the number of events by the number of simulations and the number of replications gives the total number of events generated in parallel during the course of the simulation. Table 4.12 provides the computational wall clock time required to generate a single event (either real or null).

Total Events	Replications	Event Synchronous	Time Synchronous
200,000	1	4.20 E-06 sec/event	2.77 E-06 sec/event
6,000,000	30	9.81 E-07 sec/event	5.87 E-07 sec/event
12,000,000	60	8.01 E-07 sec/event	5.04 E-07 sec/event
18,000,000	90	7.30 E-07 sec/event	5.18 E-07 sec/event
24,000,000	120	5.52 E-07 sec/event	3.88 E-07 sec/event
30,000,000	150	7.60 E-07 sec/event	5.52 E-07 sec/event
36,000,000	180	6.34 E-07 sec/event	4.61 E-07 sec/event
42,000,000	210	6.05 E-07 sec/event	4.36 E-07 sec/event

Table 4.24 - Back End Event Generation Rate Comparison.

Figure 4.25 provides a plot of the values from Table 4.24. Note the substantial gain in event generation time as the number of simultaneous replications is increased abov' 1 indicating more efficient use of the Connection Machine hardware.

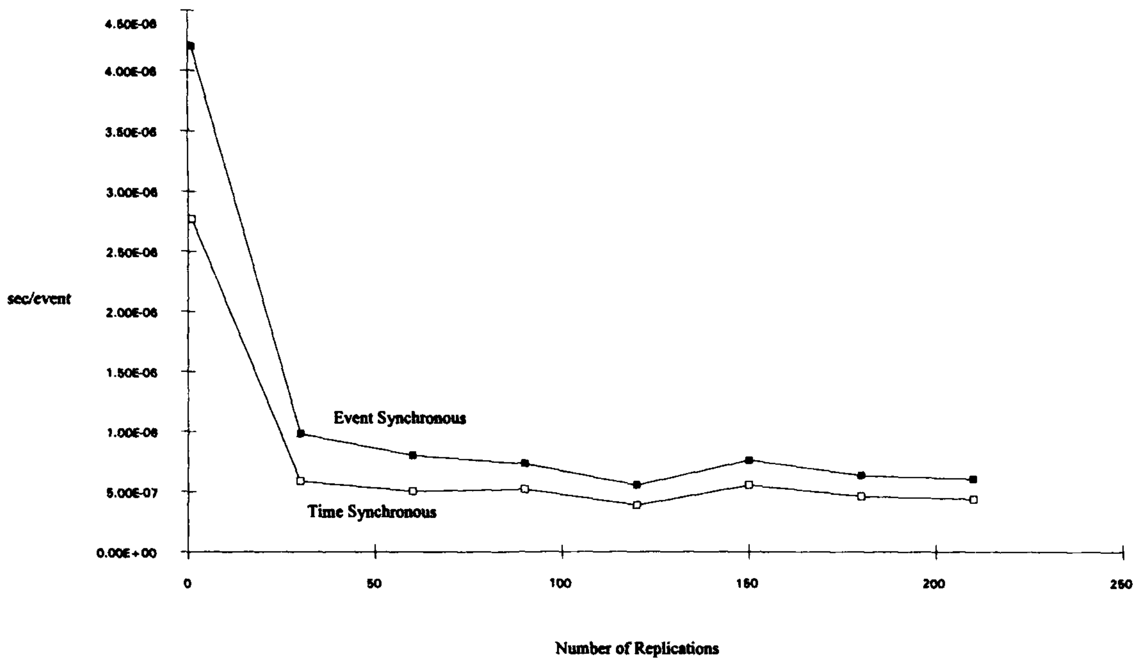


Figure 4.25 - Event Generation Rate Evolution.

Plotting the data shows a definite linear trend between the event generation rates. Note the event rate change between 90 and 120 replications as well as 150 to 180 replications. "Because some CM instructions are pipelined, the time required to perform an operation increases sub linearly with the VP ratio, so application programs may achieve better performance at higher rather than lower VP ratios."[8] It appears as if the pipelining does allow the algorithms to perform better at higher ratios to a certain point. Then, the advantage gained appears to be lost until the pipelining effect once again has a chance to improve instruction efficiency.

Overall, the time synchronous methods generate events (both real and null) approximately 1.5 times faster than the event synchronous algorithm. This average time difference is due to the algorithmic difference of two additional steps to check to see if local systems are in state 0 or k.

[8] Thinking Machines Corporation, *CM Fortran Reference Manual*, Thinking Machines Corporation, Cambridge, MA, 1991, pg. 4.

4.6.2 Front End Marking Timing.

The principle difference in generating and marking events on the front end computer is the fact that an additional random number must be generated for each event. Event synchronous simulations can also be generated using front end methods but would require major algorithmic changes. Events would be generated much slower than the front end time synchronous because of the need for local interevent time computation as a function of local system state. Front end techniques then only apply to time synchronous implementations. The following table provides the real time event generation rate for front end marking.

Total Events	Replications	Time Synchronous
200,000	1	0.94 E-06 sec/event
6,000,000	30	1.03 E-06 sec/event
12,000,000	60	1.11 E-06 sec/event
18,000,000	90	1.20 E-06 sec/event
24,000,000	120	1.20 E-06 sec/event
30,000,000	150	1.18 E-06 sec/event
36,000,000	180	1.41 E-06 sec/event
42,000,000	210	1.43 E-06 sec/event

Table 4.25 - Front End Event Generation Rates.

As the number of serial replications is increased, the only algorithmic change is the size of a serial array located at every processor to store raw replication statistical data for final mean and variance computations during the WRAP-UP phase. The following figure plots the front end event generation rate as a function of the number serial replications.

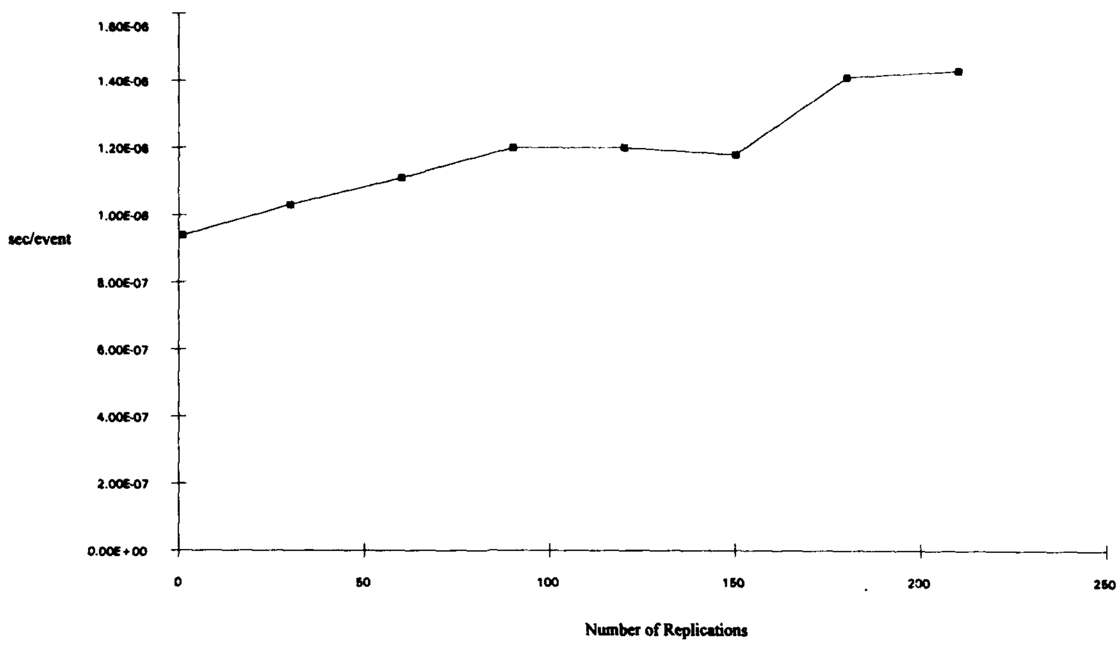


Figure 4.26 - Front End Event Generation Rates.

There is a slight overall increase in the event generation rate. This may be due to the increased size of the local array for storage of intermediate results for final statistical computations. The following figure compares the two different event marking techniques and hypothesizes possible event generation rate trends. Key is the fact that if only one replication is to be performed, use the front end, otherwise back end marking is faster.

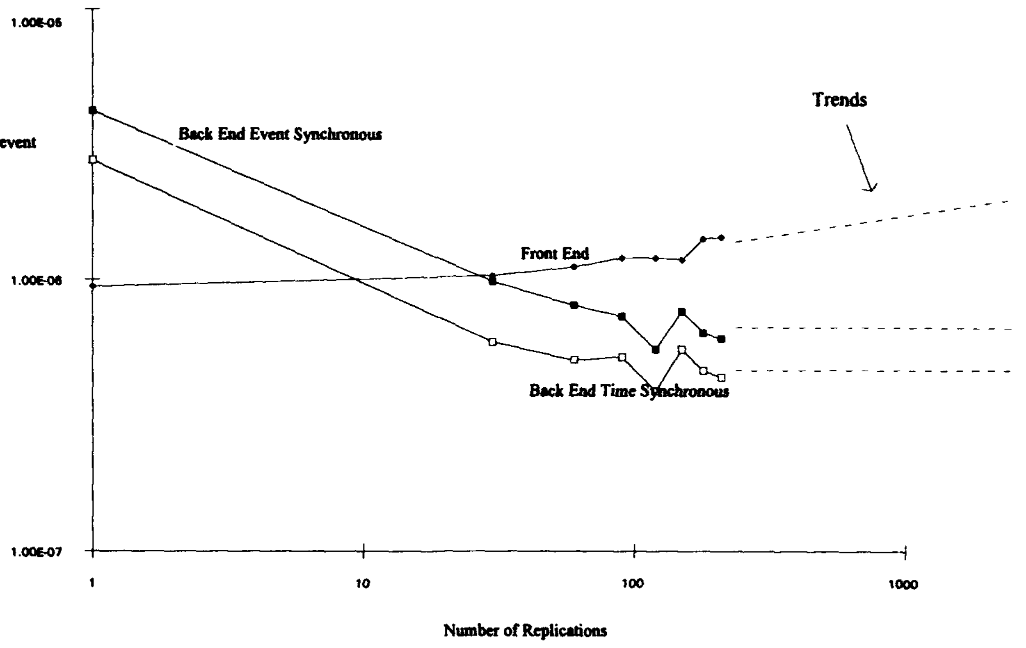


Figure 4.27 - Comparison of Front End and Back End Event Generation Rates.

4.6.3 Front and Back End Serial Replication Comparison.

Even though the SIMD architecture allows the opportunity to perform multiple replications simultaneously, there may be memory considerations if a large system is to be simulated. For this case, it may be beneficial to determine if serial back end marked replications have faster event generation rates than the equivalent front end marked simulations. For back end marking, interevent time computation as well as event marking still are conducted on the back end processors for event synchronous whereas for the time synchronous method only event marking is required at the parallel processors. Interevent times could be stored on the front end and broadcast to the processors for raw performance statistics computation. A set of time synchronous back end simulations were performed of a 10 by 20 parametric family of M/M/1/k queueing systems for 30 serial replications. The following table compares the front end times previously generated with the back end event generation times. Obviously it is much more beneficial to perform serial replications on the front end if possible.

	Event Generation Rate
Front End	0.94 E-06 sec/event
Back End	15.52 E-06 sec/event

Table 4.26 - Serial Replication Event Generation Rate Comparison.

If the simulation effort requires the generation of multiple replications and the available local processor memory is sufficient to maintain system characteristics, then use back end marking techniques. Otherwise, if only a single replication is required or local processor memory is insufficient, then front end marking seems to make the most sense.

4.6.4 Efficiency Effects.

The timing calculations given in the previous section neglect whether a real or null event was generated at any processor during the event generation phase. There is a tradeoff between time to compute a specified number of real events and the variance of a performance measure which was previously shown to be the predicted efficiency. Assume that both the time and event synchronous algorithms have the same Initialization and Wrap-up times. Define T_{Event} as the wall clock event synchronous event generation rate and T_{Time} as the time synchronous. Since we are only concerned with the generation of real events, then the real event time synchronous event generation rate is then $T_{\text{Time,real}}=T_{\text{Time}}/\bar{\varepsilon}$. Setting $T_{\text{Time,real}}=T_{\text{Event}}$ and rearranging yields:

$$\bar{\varepsilon} = \frac{T_{\text{Time}}}{T_{\text{Event}}} \tag{4.18}$$

T_{Time} and T_{Event} were measured and provided in Table 4.24 for back end marking. The following figures depict the comparison between the time synchronous *real* event generation rate and the event synchronous method's rate as a function of average efficiency. Note that there is no change in the event synchronous method's rate as it always operates at 100% efficiency.

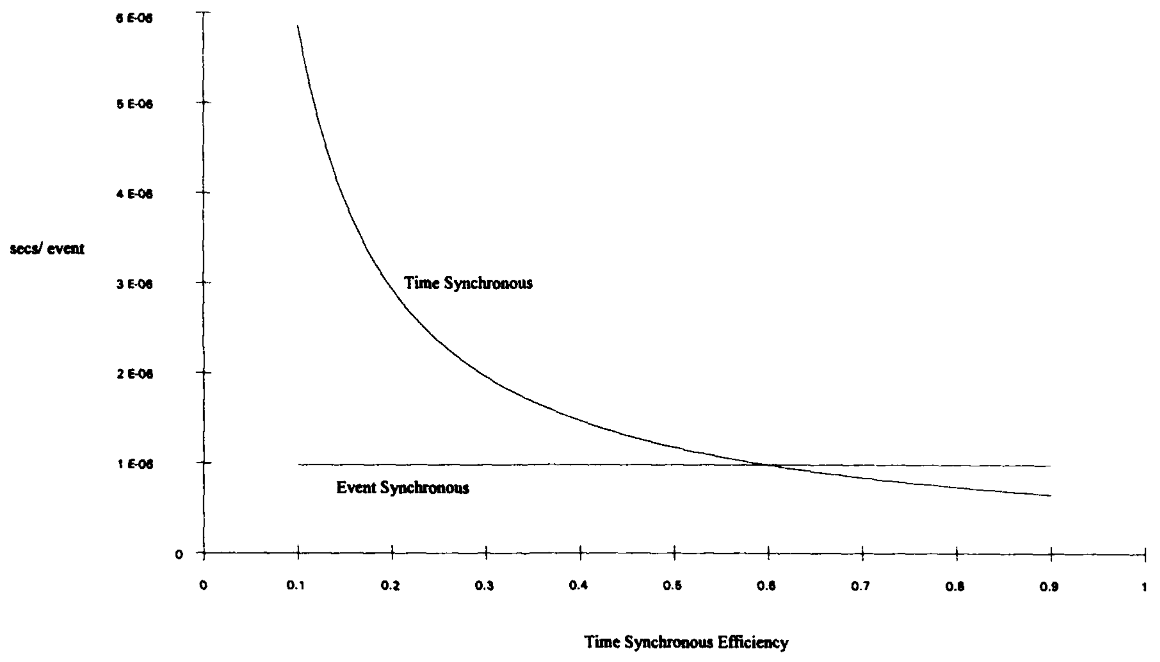


Figure 4.28 - Event Generation Coupled with Efficiency, 30 Replications.

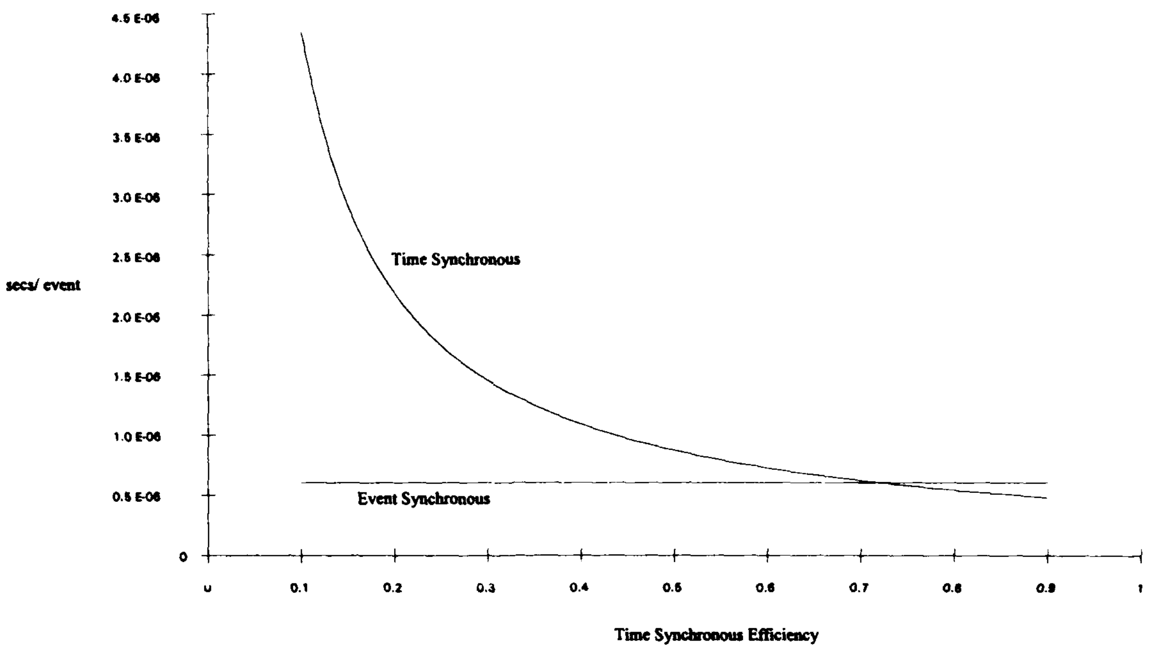


Figure 4.29 - Event Generation Coupled with Efficiency, 210 Replications.

The point where the event generation rate of the two methods crosses each other changes as a function of the number of replications. More replications translate into a higher efficiency threshold in order for the time synchronous to outperform the event synchronous in both raw clock time as well as variance of the performance measure. Define the efficiency crossover point as $\bar{\varepsilon}_b$. Figure 4.30 plots the crossover points as a function of the number of replication planes.

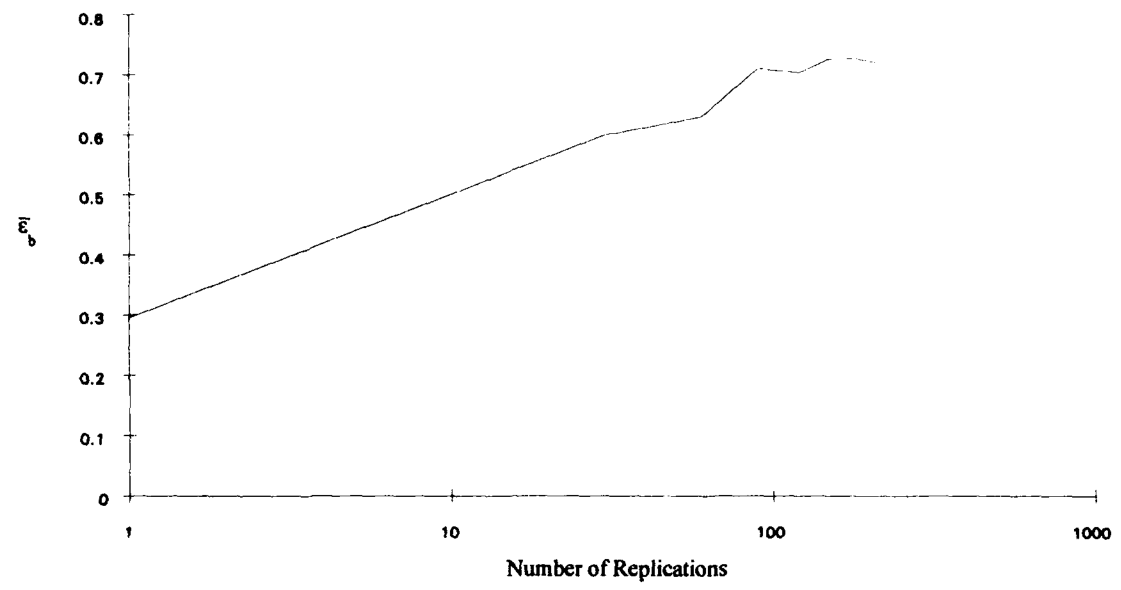


Figure 4.30 - Back End Marking Efficiency Crossover.

This figure demonstrates that as the number of replications increases, it becomes harder to justify using any time synchronous methods because of the because of the timing cost associated with generating null or infeasible events.

4.6.5 Speedup Computation.

The Gustafson-Barsis law provides a means with which to measure the achieved speedup of any application. This law required the measurement of the inherently serial component of an parallel application program and computed the speedup, S, as a function of the serial and parallel components as well as the efficiency of an application.

$$S = \beta + N(1-\beta)\bar{\varepsilon} \tag{4.19}$$

The following table depicts the relative speedup for each of the different back end methods considered. The speedup data presented assumes multiple replications. The serial component of each algorithm is only output file opening, writing and closing.

Replications	Event Synchronous	Time Synchronous
6,000	5244.8	$0.1259 + 5244.7\,\bar{\varepsilon}$
12,000	11001.5	$0.0832 + 11001.4\,\bar{\varepsilon}$
18,000	16879.4	$0.0623 + 16879.3\,\bar{\varepsilon}$
24,000	22521.8	$0.0616 + 22521.7\,\bar{\varepsilon}$
30,000	28905.4	$0.0365 + 28905.4\,\bar{\varepsilon}$
36,000	34690.1	$0.0364 + 34690.1\,\bar{\varepsilon}$
42,000	40624.4	$0.0328 + 40624.4\,\bar{\varepsilon}$

Table 4.27 - Speedup.

Even though the time synchronous generates events at a faster rate than the event synchronous, when it comes to speedup comparisons, the only significant difference is the efficiency of the selected time synchronous method. The relatively small magnitude of the inherently serial proportion of code associated with these algorithms has little effect. Because of the parallel scaleup associated with these algorithms, as the number of interevents increases, so does the speedup.

Number of Events	Number of Replications	Event Synchronous Speedup	Time Synchronous Speedup
1,000	6,000	5244.8	$0.1259 + 5244.7\,\bar{\varepsilon}$
10,000	6,000	5895.9	$0.0174 + 5895.9\,\bar{\varepsilon}$

Table 4.28 - Increased Event Speedup.

Efficiency aside, it is obvious then that the computation of speedup does not provide any more insight into the relative simulation performance of the two event generation methods.

5.0 Networks.

Chapter 5.0 extends the parametric parallel simulation techniques developed for single server queueing systems to networks of Markovian queues. Key to the extension to networks is the use of the Alias method for both front and back end marking. Even though no actual parallel simulations of these networks are performed, wall clock event generation rates should be very close to those of the M/M/1/k for any of the time synchronous methods. Event synchronous event generation rates will increase as a function of the complexity of the network because of the additional state checks required.

5.1 Background.

Consider the following network which consists of two M/M/1 queues in which customers arrive into the system with rate λ_1, are served and then depart server 1 to immediately join queue 2. After service at server 2, they depart the system. No customers are lost to this system as each queue has infinite capacity

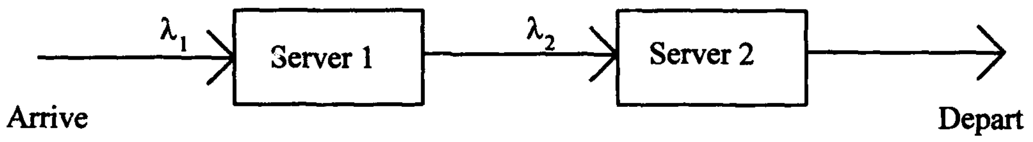


Figure 5.1 - Tandem Network.

Each server within this network has a service rate, μ_i, associated with it. However, customers can not be physically served any faster than there are customers in the system. Therefore, the rate out of server 1 is 0 when that server is in state 0 and μ_i otherwise. The effective departure rate from server 1 to queue 2 and hence queue 2's arrival rate in steady state is then:

P(Server 1 is in State 0) * Departure Rate + P(Server 1 $\neq$ State 0) * Departure Rate

Substituting the appropriate probabilities and departure rates for a M/M/1 queue yields:

$$(1-\rho_1)*0+\rho_1\mu_1=\lambda_1 \tag{5.1}$$

The arrival rate into queue 2 is actually same as the arrival rate into server 1.

Consider now a generalized network with *k* servers. Customers arrive from outside the system to each Server i, i=1..k, according to a Poisson process with rate r_i. They then join queue i until their turn in service. Once the customer completes service at i, he then joins the queue in front of Server j, j=1..k, with probability P_{ij} such that $\sum_{j=1}^{k} P_{ij} \leq 1$, or departs the system from Server i with probability $P_{ie}=1-\sum_{j=1}^{k} P_{ij}$. This queueing network is commonly referred to as a *Jackson Network.*

The actual arrival rates at any server within the network are then characterized by the following flow conservation equations:

$$\lambda_j=r_j+\sum_{i=1}^{k} P_{ij}\lambda_i,\ i=1..k \tag{5.2}$$

$P_{ij}\lambda_i$ is then the arrival rate to queue j of those departing server i.

Figure 5.2 depicts a two server Jackson Network with all possible customer ro[illegible]s. The values shown are the customer flow rates through that particular leg of the network. For subsequent discussions, the individual queues within a queueing network will be referred to as nodes.

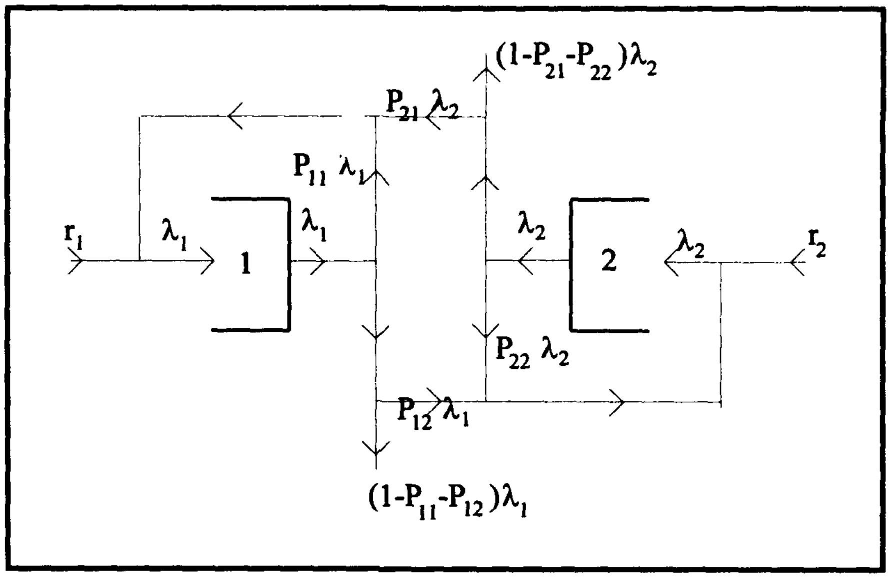


Figure 5.2 - Two Node Jackson Network.

Most open Markovian networks can be described as some derivation of a Jackson Network. A probability routing matrix, **P**, describes the relative probability that a customer may take a particular route after he has entered the system. If any P_{ij}=0, then that possible customer routing does not exist. If **P** = **0**, then the network would consist of independent queueing systems with only external arrivals to any node and all customers departing their respective systems upon completion of service.

5.2 Network Parameterization.

Parameterization of Jackson Networks for simulation across parallel processors can take many forms depending on the complexity of the systems to be simulated. Consider the following equations that describe the two node Jackson Network pictured in Figure 5.2.

Flow Conservation Equations:
$\lambda_1 = r_1 + P_{11}\lambda_1 + P_{21}\lambda_2$ $\lambda_2 = r_2 + P_{12}\lambda_1 + P_{22}\lambda_2$
Law of Total Probability:
$P_{11} + P_{12} + P_{1e} = 1$ $P_{21} + P_{22} + P_{2e} = 1$ $0 \le P_{ij}, P_{ie} \le 1$
Server Utilization:
$\lambda_1 \le \mu_1$ $\lambda_2 \le \mu_2$
Departure Probabilities:
$P_{1e} + P_{2e} > 0$

Table 5.1 - Two Node Jackson Network System Equations.

For this simple network, there are 4 equalities and 12 unknowns along with 9 inequalities or constraints. Obviously a certain number of aspects of the system in question must be known beforehand or specified before the entire network's parameters can be determined. Also, not all combinations of parametric variations may be feasible. Specifically, the node service rates, μ_i, have to be greater than the aggregate arrival rates, λ_i, otherwise, the network would tend to fill up at that particular node. Also, at least one of the external departure probabilities must be greater than zero, otherwise customers would enter the system and never be able to depart.

There are several methods with which to establish parametric variations across multiple processors. If the probability routing matrix is known, then it is possible to solve for the minimum node service rates $\mu_{i,min}$ by substituting then into the flow conservation equations in place of the λ_i's. The minimum service rates could then be solved for in terms of the external arrival rates, r_i. Conversely, if the service rates are known, then the maximum external arrival rates could be solved for.

5.2.1 Network Parameterization Option 1.

Consider the Two Node Jackson Network. For this example, let $P_{ij} = P_{ie}$; i,j = 1,2, so that customers are equally likely to either leave the system or join one of the two queues upon completion of service. Now there are 4 parameters, μ_1, μ_2, r_1, r_2, using the 2 flow equations, upon which variations can be generated. If each is varied over only 5 values, then there are 1024 total possible combinations. The following initialization algorithm provides the basic parameterization across local processors and ensures that the generated set of network characteristics are feasible.

```
        Initialize counter.
LOOP:   Determine next set of $r_i$'s and $\mu_i$'s.
        Solve for $\lambda_i$'s using flow conservation equations.
        If $\mu_i \geq \lambda_i \forall i$ Go to MOVE.
        Else, *infeasible set* - Go To LOOP.
MOVE:   Increment counter.
        All Sets Generated? If so, GO TO STOP.
        GO TO LOOP.
STOP:   Continue with Simulation.
```

5.2.2 Network Parameterization Option 2.

If the probability routing matrix is not known, but the external arrival and service rates are, then a feasible range of probabilities must be developed in order to satisfy all of the network constraints. For the two node example, the set of unknowns then reduce to 6. The one item that would be known about the routing matrix is if any of the routing *legs* are non-existent, i.e., if any $P_{ij}=0$, which would subsequently reduce the number of unknowns. The flow conservation equ[illegible] can be rearranged and solved for in terms of the individual arrival rates.

$$\lambda_1 = \frac{r_2(1-P_{11}) + r_1 P_{12}}{1 - P_{11} - P_{22} + P_{11}P_{22} - P_{12}P_{21}} \tag{5.3}$$

$$\lambda_2 = \frac{r_1(1-P_{22})+r_2P_{21}}{1-P_{11}-P_{22}+P_{11}P_{22}-P_{12}P_{21}} \tag{5.4}$$

Since $\mu_i \geq \lambda_i$, Equations 5.3 and 5.4 become:

$$\mu_1 \geq \frac{r_2(1-P_{11})+r_1P_{12}}{1-P_{11}-P_{22}+P_{11}P_{22}-P_{12}P_{21}} \tag{5.5}$$

$$\mu_2 \geq \frac{r_1(1-P_{22})+r_2P_{21}}{1-P_{11}-P_{22}+P_{11}P_{22}-P_{12}P_{21}} \tag{5.6}$$

Equations 5.5 and 5.6 must be satisfied during the course of the parameterization across the processors such that all of the simulated systems are feasible and do not violate any of the network constraints. There are two extreme cases. The first is when every $P_{ij} = 0$. As a custome [illegible] letes service at node i, he immediately exits the system from that node, not much of a network. The other extreme set comes when the inequalities in Equations 5.5 and 5.6 are replaced by equalities. Equations 5.7 and 5.8 represent the simplified equations for the case where $\lambda_i = \mu_i \forall i$.

$$P_{11} = \frac{\mu_1 - \mu_1 P_{21} + r_1}{\mu_2} \tag{5.7}$$

$$P_{12} = \frac{\mu_1 - \mu_1 P_{22} - r_2}{\mu_2} \tag{5.8}$$

The following initialization algorithm provides the basic parameterization across the local processors and ensures that the generated parameterization set is feasible.

	Initialize counter.
LOOP:	Determine next set of P_{ij}'s.
	Solve for λ_i's using Eqns. 5.3 and 5.4 and others.
	If $\mu_i \geq \lambda_i \forall i$ Go to MOVE:
	Else, *infeasible set* - Go To LOOP.
MOVE:	Increment counter.
	All Sets Generated? If so, GO TO STOP.
	GO TO LOOP.
STOP:	Continue with Simulation.

Some other constraints could be introduced so that the family of networks can be articulated in greater detail to facilitate performance comparisons. One possible consideration is to set the departure rates, D, at any node, $D_i = P_{ie}\lambda_i$. The last obvious and most complicated paramaterization scheme involves any manner of combining the two previous methods and will be not be discussed.

5.3 Simulation Details.

The number of customers at each of the k nodes in the general Jackson Network is independent and is equivalent to a simple M/M/1 queue.

$$P(\text{n customers at node j}) = \left(\frac{\lambda_j}{\mu_j}\right)^n \left(1 - \frac{\lambda_j}{\mu_j}\right) \tag{5.9}$$

The joint customer distribution is given by:

$$P(n_1, n_2, \ldots, n_k) = \prod_{j=1}^{k} \left(\frac{\lambda_j}{\mu_j}\right)^{n_j} \left(1 - \frac{\lambda_j}{\mu_j}\right) \tag{5.10}$$

The above equation describes the number of customers at server i as the same as a M/M/1 queueing system with rates λ_i and μ_i. However, the arrival process at each of the nodes within the system is no longer Poisson as there is a possibility that a customer will be fed back to the node he just departed. "When feedback is allowed, the steady-state probabilities of the number at any given station have the same distribution as in an M/M/1 model even though the model is not M/M/1"[9]. Ross' conclusion, based on *Jackson's Theorem*, then provides the basis for SIMD parallel simulation of Jackson Networks.

There are two types of events within this network. The first is an external arrival from outside the system. A departure from node i bound for node j is the second. For the simple two node system, there are then 6 possible events. In fact for an n-node system, the total number of possible events is n(n+2). Now, based on Jackson's Theorem, system interevents can be generated like the M/M/1 with exponential interevent times. The interevent rate, q, for a single k node system is then:

$$q = \sum_{i=1}^{k} (r_i + \mu_i) \tag{5.11}$$

There are then several alternatives as to the selection of the parametric variations of any system. The first is to hold the probability routing matrix constant and vary the event generation rates across processors. The second is to maintain a constant event generation rate across all processors and vary the routing matrix. Another may be to vary both sets of parameters in any manner of combinations.

[9] S. M. Ross, *Introduction to Probability Models*, 4th Edition, Harcourt Brace Jovanovich, Boston, 1989, pg. 369.

As in the simple case of the M/M/1/k, there are two event generation methods available that map well to the SIMD architecture; event and time synchronous. The simulation of a network is now slightly more complicated in that there is a multi-dimensional state space associated with the network state. On the CM-2, the state vector would be stored as a serial dimension at every local processor and the applicable transition vector would be applied during the event generation phase of the simulation. The following are the seven different possible state transition vectors that could be applied to the state vector for the Two Node Jackson Network during state updating:

Generated Event	**Vector**
External Arrival to Node 1	(+1, 0)
External Arrival to Node 2	(0,+1)
Departure from Node 1 to Node 2	(-1,+1)
Departure from Node 2 to Node 1	(+1,-1)
System Departure from Node 1	(-1, 0)
System Departure from Node 2	(0,-1)
Departure from Node i back to Node i, i=1,2	(0, 0)

Table 5.2 - Two Node Network State Transition Vectors.

5.3.1 Front End Marking.

Time synchronous parallel simulations require front end generated interevents which are either marked on unmarked as they are broadcast to the individual processors. Consider a k node network. The marked front end interevent rate would then be generated by taking the maximum of each individual possible event across all processors.

$$q_{\max,FE} = \sum_{i=1}^{k} (r_{i,\max} + \mu_{i,\max}) \qquad (5.12)$$

Marked front end events are generated in a similar fashion as that described for the M/M/1 or M/M/1/k. Events are marked according to the following probability formulation:

$$P(\text{External Arrival @ Node i}) = \frac{r_{i,max}}{q_{max,FE}} \tag{5.13}$$

$$P(\text{Departure @ Node i} \Rightarrow \text{Join Node j}) = \frac{\mu_{i,max} P_{ij}}{q_{max,FE}} \tag{5.14}$$

Events are then accepted on the back end according to the following local processor probabilities:

$$P(\text{Accepting External Arrival @ Node i})\Big|_{\text{Processor a}} = \frac{r_{i,max}}{q_{max,FE}} \frac{r_{i,a}}{r_{i,max}} = \frac{r_{i,a}}{q_{max,FE}} \tag{5.15}$$

$$P(\text{Accepting Departure @ Node i} \Rightarrow \text{Join Node j})\Big|_{\text{Processor a}} = \frac{\mu_{i,a} P_{ij,a}}{q_{max,FE}} \tag{5.16}$$

The effective probability at which events will be discarded at the local processor is:

$$P(\text{Not Accepting Event})\Big|_{\text{Processor a}} = \frac{q_{max,FE} - \sum_{i=1}^{k} (r_{i,a} + \mu_{i,a})}{q_{max,FE}} \tag{5.17}$$

Interevent times are generated in the same manner as the simple queueing systems but now use the network interevent rate.

$$\Delta\tau = -\frac{1}{q_{\max,FE}}\ln(1-U) \qquad (5.18)$$

5.3.2 Back End Marking.

The back end marking technique would generate interevents at a lower rate, that of the maximal rate system. In comparison, the interevent rate of the front end marking method may not even correspond to any of the systems being simulated.

$$q_{\max,BE} = \mathrm{MAX}\left(\sum_{i=1}^{k}(r_i+\mu_i)\right)\Bigg|_{\text{All Processors}} \qquad (5.19)$$

The unmarked events are sent to the local processors and typed according to the following set of probabilities.

$$P(\text{Accepting External Arrival @ Node i })\big|_{\text{Processor a}} = \frac{r_{i,a}}{q_{\max,BE}} \qquad (5.20)$$

$$P(\text{Accepting Departure @ Node i} \Rightarrow \text{Join Node j})\big|_{\text{Processor a}} = \frac{\mu_{i,a}P_{ij,a}}{q_{\max,BE}} \qquad (5.21)$$

$$P(\text{Null Event})\big|_{\text{Processor a}} = 1 - \frac{\sum_{i=1}^{k}(r_i+\mu_i)\Big|_{\text{Processor a}}}{\mathrm{MAX}\left(\sum_{i=1}^{k}(r_i+\mu_i)\right)\Bigg|_{\text{All Processors}}} \qquad (5.22)$$

The primary advantage to back end marking is that local processor event rates can be manipulated across the range of systems to minimize or cancel the Null event effect. This effect can not be avoided using the front end marking/ standard clock technique.

Therefore, by equating the back end event generation rate across every local processor, the Null event effect no longer occurs. Equation 5.23 then describes essentially a required additional constraint pertaining to event rate parameterization and added to Table 5.1.

$$\sum_{i=1}^{k}(r_i + \mu_i)\Big|_{\text{Processor a}} = \text{MAX}\left(\sum_{i=1}^{k}(r_i + \mu_i)\right)\Bigg|_{\text{All Processors}} \tag{5.23}$$

5.3.3 Event Synchronous Simulations.

Event synchronous network simulations are performed in exactly the same manner as they were for the M/M/1/k system. When the M/M/1/k system was in either state 0 or state k, the next event was either an arrival or a departure, respectively. Now, if any of the network nodes are in state 0, any of the possible service/ departure events from that set of nodes in state 0, must be precluded from being generated. The set of possible transition events with their associated vector is then a function of the current state of the network at that particular instant in time. Once again an indicator variable I_i is used to adjust the interevent generation rate.

$$q_{\text{Event}} = \sum_{i=1}^{k}(r_i + \mu_i I_i);\ I_i = 1 \text{ iff } n_i \neq 0 \tag{5.24}$$

Equation 5.24 is then used to compute the interevent times. Events are marked with the following probabilities at the local processors:

$$P(\text{Accepting External Arrival @ Node i })\Big|_{\text{Processor a}} = \frac{r_{i,a}}{q_{\text{Event}}} \tag{5.25}$$

$$P(\text{Accepting Departure @ Node i} \Rightarrow \text{Join Node j})\Big|_{\text{Processor a}} = \frac{\mu_{i,a} P_{ij,a}}{q_{\text{Event}}} \qquad (5.26)$$

As in the simple queueing system, the network event synchronous generation method operates at 100% efficiency. Interevent time generation is now much more complicated because of all of the possible combinations of nodes which may be in state 0 at any instant during the simulation.

5.4 Network Efficiency Across Processors.

The proportional variance of a performance measure between the event synchronous and time synchronous methods was previously shown to be the processor efficiency of the time synchronous method. Efficiency calculations must now take into account inefficient events for the Jackson Network. An attempted departure from any node when that particular node is in state 0 is the possible inefficient event.

Processor efficiency has also been shown to be a function of the parameterization scheme. For Markovian networks, routing matrix variations will affect the resulting state 0 (or k) effects at each node of the simulated system whereas parametric variations of *real* events (either external arrivals or service completions) affect local null event generation.

The probability of a Null event now coupled with this set of possible state 0 effects is:

$$P(\text{Null}_0) = \sum_{i=1}^{k}(P(\text{Departure @ Node i} \cap (n_i = 0) \cap \overline{\text{Null}}) + P(\text{Null})) \qquad (5.27)$$

The probability of any node, i, being in state 0 is just:

$$P(\text{Node i in State 0}) = 1 - \frac{\hat{\lambda}_i}{\mu_i} \qquad (5.28)$$

The probability of generating a departure from any node, regardless of the event generation method is:

$$P(\text{Departure from Node } i) = \frac{\mu_i}{q_{\max}} \tag{5.29}$$

5.4.1 Front End Marking Efficiency.

Taking the relationship developed in Equation 5.27, and applying Equations 5.28 and 5.29 for the state 0 effect and Equation 5.21 for the Null event effect, the time synchronous front end marking inefficiency at processor a is then:

$$P(\text{Null}_0)\Big|_{\text{Processor a}} = \sum_{i=1}^{k}\left(\left(1-\frac{\lambda_{i,a}}{\mu_{i,a}}\right)\frac{\mu_{i,a}}{q_{\max,FE}}\right) + \left(1-\frac{\sum_{i=1}^{k}(r_{i,a}+\mu_{i,a})}{q_{\max,FE}}\right) \tag{5.30}$$

Algebraically rearranging Equation 5.30 yields:

$$P(\text{Null}_0)\Big|_{\text{Processor a}} = 1-\frac{1}{q_{\max,FE}}\sum_{i=1}^{k}(r_{i,a}+\lambda_{i,a}) \tag{5.31}$$

Efficiency, ε, is defined once again as 1-P(Null). By substituting the definition of the maximum interevent rate given by Equation 5.12, processor efficiency is then:

$$\varepsilon(\text{Front End})\Big|_{\text{Processor a}} = \frac{\sum_{i=1}^{k}(r_{i,a}+\lambda_{i,a})}{\sum_{i=1}^{k}(r_{i,\max}+\mu_{i,\max})} \tag{5.32}$$

Overall average efficiency, $\bar{\varepsilon}$, across N processors, is computed in the same manner as that of the simple queueing systems. Conventional averaging is used rather than integration because of the discrete nature of the parameterization.

$$\bar{\varepsilon}(\text{Front End}) = \frac{1}{N}\sum_{a=1}^{N}\left(\frac{\sum_{i=1}^{k}(r_{i,a} + \lambda_{i,a})}{\sum_{i=1}^{k}(r_{i,\max} + \mu_{i,\max})}\right) \tag{5.33}$$

5.4.2 Network Back End Marking Efficiency.

Recall that the back end generation method can be parameterized such that no Null events are generated. The state 0 effect probability can then be taken directly from the first term of Equation 5.30. Rearranging that term yields:

$$P(\text{Null}_0)\Big|_{\text{Processor a}} = \sum_{i=1}^{k}\left(\left(1-\frac{\lambda_{i,a}}{\mu_{i,a}}\right)\frac{\mu_{i,a}}{q_{\max,FE}}\right) = \sum_{i=1}^{k}\left(\frac{\mu_{i,a}-\lambda_{i,a}}{q_{\max,BE}}\right) = \sum_{i=1}^{k}\left(\frac{\mu_{i,a}-\lambda_{i,a}}{\sum_{j=1}^{k}(r_{j,a}+\mu_{j,a})}\right) \tag{5.34}$$

Individual processor efficiency can also be computed by subtracting Equation 5.34 from 1. Average processor efficiency across N processors then is:

$$\bar{\varepsilon}(\text{Back End}) = \frac{1}{N}\sum_{a=1}^{N}\left(1-\sum_{i=1}^{k}\left(\frac{\mu_{i,a}-\lambda_{i,a}}{\sum_{j=1}^{k}(r_{j,a}+\mu_{j,a})}\right)\right) \tag{5.35}$$

5.5 Buffered Networks.

Many existing systems can be simulated in parallel using the techniques developed for Jackson Networks. However, real world systems which can be modeled using these techniques usually have some sort of buffer capacity at each of their nodes. Extending the SIMD parallel simulation concepts developed for basic Jackson Networks to networks with buffered nodes is similar to the differences between the single M/M/1 and M/M/1/k queueing systems.

Once again consider a general Jackson Network as detailed in Section 5.1. Instead of infinite capacity, there is now a finite buffer capacity at each node. The flow conservation equations now have a additional factor that accounts for thinning that occurs as a result of a full buffer at any node. Since each node in the network is actually a M/M/1/k queue, the limiting probabilities are known. The limiting probability of node j being in state i is then:

$$\pi(i_j) = \frac{1 - \frac{\lambda_{i,j}}{\mu_{i,j}}}{1 - \left(\frac{\lambda_{i,j}}{\mu_{i,j}}\right)^{k+1}} \left(\frac{\lambda_{i,j}}{\mu_{i,j}}\right)^{i} \tag{5.36}$$

The actual departure rate from node j is then computed in the same fashion as it was for the non-buffered network. The departure rate from node j is 0 when that node is in state 0 and μ_j otherwise.

$$\text{Departure Rate from Node } j = 0 * \pi(0_j) + \mu_j(1 - \pi(0_j))$$

The arrival rate to any node and thus its effective departure rate is then a function of the amount of time that the individual node is at or below buffer capacity. If K_j is the buffer capacity of node j, then $1-\pi(K_j)$ is the long run probability of node j being able to accept arrivals from any source, either externally or from other nodes. After some algebraic manipulation, the effective departure rate is given by Equation 5.37 which is expressed in terms of the node's arrival rate.

$$\mu_j(1-\pi(0_j)) = \lambda_j(1-\pi(K_j)) \tag{5.37}$$

The flow conservation equations for the network then become:

$$\lambda_i = r_i + \sum_{j=1}^{k} \lambda_j(1-\pi(K_j))P_{ji} \tag{5.38}$$

Interevent times are generated in the same way they were for the non buffered network with the rate a function of the external arrival and appropriate service rates whether time or event synchronous is chosen.

There are now two additional types of inefficient network events that can occur. The first is an attempted external arrival to a node which is full. If this particular event is generated, the customer is essentially sent away and lost to the system. The second inefficient event is an attempted departure from node i to join node j when node j is full. This event has possibly dire consequences from a simulation standpoint. If the attempted event is treated by applying no state change, i.e. leaving the customer at server i, then there is the possibility of network deadlock as more customers join the system. If the attempted internal arrival to a full queue is rerouted to externally exit the system, then the possibility of deadlock can be avoided.

As before, implementing the event synchronous method would avoid the attempted generation of an inefficient event.

5.6 Network Performance Measures.

Once the chosen method of generating the parametric variations has been selected, multi-dimensional surfaces can be generated of several performance measures that predict the long run performance of the system in steady state. As with the case of the M/M/1 queue, the average aggregate queue length for the network can also be determined. Recall that the average queue length for the M/M/1 queue can be ascertained by the equation, $\overline{L} = \frac{\lambda}{\mu - \lambda}$. Extending this equation to Jackson Networks involves computing the average queue length at each node.

$$\overline{L} = \sum_{i=1}^{j} \frac{\lambda_i}{\mu_i - \lambda_i} \tag{5.39}$$

Recall that the shape of the parametric variance of the M/M/1 queue can be also be determined by using the equation, $\sigma_L^2 = \sum_{i=1}^{\infty} (i - \overline{L})^2 P_i$. In a similar manner as that for the queue length, , the variance of the k node Jackson queueing network is just the sum of the variances of the individual nodes assuming independence of each node.

$$\sigma_L^2 = \sum_{j=1}^{k} \sum_{i=1}^{\infty} (i - \overline{L})^2 P_{i,\text{node } j} \tag{5.40}$$

Substituting Equation 5.39 for $\overline{L}$ in Equation 5.40 yields:

$$\sigma_L^2 = \sum_{j=1}^{k} \sum_{i=1}^{\infty} \left(i - \sum_{h=1}^{j} \frac{\lambda_h}{\mu_h - \lambda_h} \right)^2 \left(\frac{\lambda_j}{\mu_j} \right)^i \left(1 - \frac{\lambda_j}{\mu_j} \right) \tag{5.41}$$

5.7 Predicted Simulation Performance.

No parallel simulations of Jackson Networks were executed. However, two observations can be made. The wall clock event generation time for either front or back end marking of time synchronous simulations should be roughly comparable to those achieved for the M/M/1/k queueing systems. Regardless of the number of possible events, the Alias method still only requires two instructions to determine the next event type. Some increase in timing may be observed similar to the increase experienced with front end serial replications as the number of network nodes increase, as a result of the local processor memory requirement for storage of state data.

Event synchronous simulations on the other hand may show substantial increases in wall clock event generation rates. Every node's state must be checked prior to interevent computation and state updating. For infinite capacity, k node networks, this would translate into k-1 additional instructions when compared with the M/M/1 queueing system. For the finite buffer capacity case, 2(k-1) additional parallel instructions are required. These additional instructions would translate into increased event generation rates.

6.0 Conclusions and Research Contributions.

There has been increasing interest in performing massively parallel simulations of parametric families of queueing systems over the past few years. Driving this interest was the development by Vakili of the Standard Clock algorithm. His standard clock algorithm makes use of the uniformization procedure to ensure that all system parametric variations maintain the same system time throughout the course of the simulation. This thesis focused on the various means with which M/M/1 and M/M/1/k queueing systems can be parameterized and simulated on a SIMD supercomputer. It detailed the advantages and disadvantages for the differing algorithms. It also provided a theoretical means with which to compute relative variance performance of the different parallel simulation techniques. Extensions of the different methods to Jackson Networks were also provided and showed that the techniques developed for single server queueing systems readily scale to networks of Markovian queues.

6.1 SIMD Parallel Simulation Techniques.

The Single Instruction Multiple Data architecture places unique requirements on any application program in order for it to run. Every parallel processor accepts and executes as appropriate, centrally issued commands from the front end computer. Vakili's standard clock algorithm stipulated that a marked Poisson process be generated by the front end computer and then the marked events accepted or rejected locally by the individual parallel processors. A major topic of this thesis is the comparison of Vakili's algorithm with two alternatives. The first alternative was called event synchronous. Rather than generate an uniformized process, the event synchronous algorithm allows each parallel processor to generate the next event based on local system state stored at each processor during each event generation phase of the simulation. The second alternative was the time synchronous back end marking method and was developed as the means with which to actually perform the event synchronous method. Rather than generate a marked event, the front end directs each local processor to generate and mark its own event in accordance with local system probabilities. It was shown that there are cases when either of these two alternative methods are more beneficial to use than the standard clock for parallel simulations.

6.1.1 Event and Time Synchronous Parallel Simulation Techniques.

There are distinct advantages and disadvantages to both the time synchronous and event synchronous parallel simulation algorithms. Selection of the algorithm to use is first based on the goals of the simulation and the developed terminating criteria for the parametric variations. Are the range of systems to be compared after each has reached a certain system time or after a certain number of events have been generated?

A distinct advantage to the event synchronous is that every event generated is used at every parallel processor. Disadvantages include the fact that every processor has an unique system time and that it also requires additional parallel state checking instructions to generate the next event. Thus, even though this method uses all events, it does produce events at a slower wall clock rate.

A significant advantage to the time synchronous method is implied by its name. Another advantage is that it generates events at a faster wall clock rate than the event synchronous method. The principal disadvantage is that selection of the parameterization across the available processors directly affects the efficient use of those processors. Some parameterization schemes are better than others. Overall efficient use of the available processors is a function of the selection and range of the parameterization.

6.1.2 Front End vrs. Back End Marked Event Generation.

The standard clock method generates a marked event at the front end which is subsequently broadcast to all parallel processors for subsequent acceptance or rejection based on local parameters. For a M/M/1 or M/M/1/k queueing system, this requires the generation of three random numbers to 1) generate the interevent time, 2) mark the event and 3) accept or reject the event locally. An equivalent but more efficient technique requires the generation of only two random numbers is to have the events marked on the back end. There are advantages and disadvantages to each also.

Front end marking generates events faster than back end marking if only a single replication of the simulation is performed. If multiple replications are required, then back end marking is faster as it exploits the SIMD architecture.

There is also a difference in the time synchronous efficiency between the front and back end methods. Because events must be generated at a rate of the summation of the maximum of all of the parameters for front end marking, there is unavoidable thinning that will occur. Selection of the parameterization scheme using back end methods minimizes or negates any thinning or null event effects.

6.1.3 The Parallel Alias Method.

In order to perform any of the back end marking techniques, to include the event synchronous method, it was necessary to develop an algorithm that efficiently generates the next event during the course of a simulation with the appropriate rate. The Alias method for generating discrete distributions is ideally suited for the SIMD architecture because each processor is essentially identified as an array index. One of the contributions of this research was the development and implementation of a parallel Alias method which set up the two required alias arrays at every parallel processor for use during simulations requiring back end marking. The principle advantage for the use of this method is now event generation can be accomplished using only two random numbers as compared to three which are required for front end marking.

6.2 Development and Use of Processor Efficiency.

The time synchronous event generation methods use uniformization techniques to essentially either throw away events or attempt to generate infeasible state transitions. There is then a theoretical percentage of generated events that is actually never used at every processor which is a function of the parameterization of the systems. Chapter 3.0 of the thesis provided the analytical derivation for the average processor efficiency of the different methods. Chapter 4.0 then showed how the ratio of any performance measure event synchronous to time synchronous variance is actually the predicted efficiency. Therefore, given a variance and either method, it is possible to predict the other method's performance measure variance.

6.3 Variance Reduction Through Stochastic Coupling.

The SIMD architecture provided a means with which to induce high degrees of correlation between successive members of a parametric family. This high correlation was then used to significantly reduce the variance of the successive differences. By significantly reducing the variance of the differences, it is then possible to more rapidly ascertain the ordinal optimality of a range of systems that it would be using cardinal optimality techniques and computing the absolute performance measures.

Another significant result was in the difference in the degree of correlation between the front end and back end marking methods. The front end was able to achieve higher levels of correlation. This was due to the fact that every processor had to either accept or reject the same event type. Back end marking allows possible different event types to occur simultaneously during a parallel simulation. By achieving this higher degree of correlation, some front end methods with lower overall efficiencies were able to outperform more efficient back end methods when comparing the variance of successive differences.

6.4 Future Research Topics and Directions.

There are many different topics and directions that researchers in techniques of parametric parallel simulations could go. This section attempts to highlight a few that have arisen during the course of the research and were not looked into with great detail.

Efficiency:

The time synchronous efficiency computations described in Chapter 3.0 assume a uniform spread of the selected parametric variation across the available processors. There may be times when a non-uniformly distributed parameterization scheme could be used. One area of future research may be the analytical formulation of average efficiency for general parameterization.

Variance Reduction Techniques:

There are a wide variety of variance reduction techniques may perform well within the framework of the SIMD architecture. Besides common random numbers, the use of Antithetic Variates seem as if it would perform reasonably well. Antithetic Variates are applicable to single systems and are based on the ability to induce negative correlation between pairs of like systems by using synchronized complementary random numbers (U and 1-U). A good discussion of this method can be found in [Law 1991]. Extending this concept to a SIMD machine is straight forward. Recall that the M/M/1/k parametric family of systems simulated in Chapter 4.0 was varied over three dimensions with the first two dimensions representing different system permutations, and the third, stochastic. Since Antithetic Variates are applied only to stochastic variants, then complementary random numbers could be generated on pairs of adjacent simulation planes to effect the theoretical variance reduction.

Non-Exponential Distributions:

This thesis touched on the method with which non-exponential distributions could be simulated in parallel through the use of its hazard rate. No parallel simulations were performed using any non-exponential distributions. Front and back end marking techniques need to be developed to fully exploit the SIMD architecture and quickly generate events.

MIMD Architectures:

The SIMD architecture is slowly being phased out and replaced with more flexible MIMD/ SIMD machines such as the CM-5. Performing parametric parallel simulations on a MIMD architecture places additional burdens on the programmer to synchronize the parallel processors. The generalized event and time synchronous approaches will have to be adjusted to work correctly on a MIMD machine. Specific implementations may be machine or memory model specific. The distributed memory model seems better suited to both front and back end marking techniques because of the small amount of interprocessor communications required until performance measure statistical compilation. This area is wide open for research.

Large Scale System Simulations:

By far, the largest area of research into parallel simulation is in the large scale distributed simulation of a large system. It is possible that sparse matrix techniques could be developed for the SIMD architecture to allow large decomposed simulations to run quickly and efficiently. However, with the eventual demise of the SIMD pure machine, this area may never be explored.

7.0 Bibliography.

[Almasi, 1989]	G. S. Almasi, and A. Gottlieb, *Highly Parallel Computing*, The Benjamin/ Cummings Publishing Co., Redwood City, California, 1989.
[Banks, 1984]	J. Banks and J. S. Carson, *Discrete-Event System Simulation*, Prentice-Hall, Englewood Cliffs, New Jersey, 1984.
[Bhavsar, 1987]	V. C. Bhavsar and J. R. Issac, "Design and Analysis of Parallel Monte Carlo Algorithms", *SIAM Journal of Scientific and Statistical Computing*, 8, (1) 1987, pp. s73-s95.
[Blanchard, 1990]	B. S. Blanchard and W. J. Fabrycky, *Systems Engineering and Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
[BFS, 1987]	P. B. Bratley, B. L. Fox and L. E. Schrage, *A Guide to Simulation*, 2nd Edition. Springer- Verlag, New York, 1987.
[Fishman, 1983]	G. S. Fishman, "Accelerated accuracy in the Simulation of Markov Chains", *Operations Research*, 31, 1983, pp. 466-487.
[Fishman, 1983]	G. S. Fishman, "Accelerated convergence in the Simulation of Countably Infinite State Markov Chains", *Operations Research*, 31, 1983, pp. 1074-1089.
[Fishman, 1978]	G. S. Fishman, *Principles of Discrete Event Simulation*, John Wiley & Sons, New York, 1978.

[Fishman, 1973]	G. S. Fishman, *Concepts and Methods in Discrete Event Digital Simulation*, John Wiley & Sons, New York, 1973.
[Fox, 1990]	B. L. Fox, "Generating Markov-Chain Transitions Quickly: I:, *ORSA Journal on Computing*, 2, (2), 1990, pp. 126-135.
[Fox, 1986]	B. L. Fox, and P. W. Glynn. "Discrete-Time Conversion for Simulating Semi-Markov Processes", *Operations Research Letters*, 5, (4), 1986, pp. 191-196.
[Fujimoto, 1990]	R. M. Fujimoto, "Parallel Discrete Event Simulation", *Communication of ACM*, 33, (10), 1990, pp. 30-53.
[Glynn, 1991]	P. W. Glynn, and P. Heidelberger, "Analysis of Parallel Replicated Simulations Under a Completion Time Constraint", *ACM Transactions on Modeling and Computer Simulations*, 1, 1991, pp. 3-23.
[Heidelberger, 1986]	P. Heidelberger, "Statistical Analysis of Parallel Simulations", *Proceedings of the 1986 Winter Simulation Conference*, pp. 290-295.
[Heidelberger, 1988]	P. Heidelberger, "Discrete Event Simulations and Parallel Processing: Statistical Properties", *SIAM Journal of Scientific and Statistical Computing*, 9, (6),1988, pp. 1114-1132.
[Heidelberger, 1991]	P. Heidelberger, and D. M. Nichol, "Simultaneous Parallel Simulations of Continuous Time Markov Chains at Multiple Parameter Settings", *Proceedings of the 1991 Winter Simulation Conference*, pp. 602-607.

[Hillis, 1985]	W. D. Hillis, *The Connection Machine*, MIT Press, Cambridge Ma., 1985.
[Hillis, 1986]	W. D. Hillis, and G. L. Steele, "Data Parallel Algorithms", *Communications of the ACM*, 29, (12), 1986, pp. 1170-1183.
[Ho, 1993]	Y. C. Ho, C. G. Cassandras, and M. Makhlouf, "Parallel simulation of real-time systems via the Standard Clock approach", *Mathematics and Computers in Simulation*, 35, 1993, pp. 33-41.
[Ho, 1988]	Y. C. Ho, S. Li, and P. Vakili, "On the Efficient Generation of Discrete Event Sample Paths under Different System Parameter Values", *Mathematics and Computers in Simulation*, 30, 1988, pp. 347-370.
[Kleinrock - I, 1975]	L. Kleinrock, *Queueing Systems Volume 1: Theory*, John Wiley & Sons, New York, 1975.
[Kleinrock - II, 1976]	L. Kleinrock, *Queueing Systems Volume 2: Computer Applications*, John Wiley & Sons, New York, 1976.
[Law, 1991]	A. M. Law, and W. D. Kelton, *Simulation Modeling and Analysis*, 2nd Edition, McGraw-Hill, New York, 1991.
[Lewis, 1992]	T. G. Lewis, H. El_Rewini, *Introduction to Parallel Computing*, Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1992.
[Mitrani, 1982]	I. Mitrani, *Simulation Techniques for Discrete Event Systems*, Cambridge University Press, Cambridge England, 1982.

[Quinn, 1990]	M. J. Quinn, and P. J. Hatcher, "Data-Parallel Programming on Multicomputers", *IEEE Software*, Sept 1990, pp. 69-76.
[Righter, 1989]	R. Righter, and J. C. Walrand, "Distributed Simulation of Discrete Event Systems", *Proceedings of the IEEE*, 77 (1), 1989, pp. 99-113.
[Ross, 1990]	S. M. Ross, *A Course in Simulation*, Macmillian, New York, 1990.
[Ross, 1989]	S. M. Ross, *Introduction to Probability Models*, 4th Edition, Harcourt Brace Jovanovich, Boston, 1989.
[Ross, 1983]	S. M. Ross, *Stochastic Processes*, John Wiley & Sons, New York, 1983.
[Sauer, 1981]	C. H. Sauer, and K. M. Chandy, *Computer Systems Performance Modeling"*, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1981.
[CMF PG 1991]	Thinking Machines Corporation, *CM Fortran Programming Guide*, Thinking Machines Corporation, Cambridge, MA, 1991.
[CMF RM 1991]	Thinking Machines Corporation, *CM Fortran Reference Manual*, Thinking Machines Corporation, Cambridge, MA, 1991.
[CMF UG 1991]	Thinking Machines Corporation, *CM Fortran User's Guide*, Thinking Machines Corporation, Cambridge, MA, 1991.

[CMF ON 1991] Thinking Machines Corporation, *CM Fortran Optimization Notes: Slicewise Model*, Thinking Machines Corporation, Cambridge, MA, 1991.

[Tucker, 1988] L. W. Tucker, and G. G. Robertson, "Architecture and Applications of the Connection Machine", *Computer*, 21(8), 1988, pp. 26-38.

[Vakili, 1991] P. Vakili, "Using a Standard Clock Technique for Efficient Simulation", *Operations Research Letters,* 10, pp. 445-452, 1991.

[Vakili, 1992] P. Vakili, "Massively Parallel and Distributed Simulation of a Class of Discrete Event Systems: A Different Perspective", *ACM Transactions on Modeling and Computer Simulation*, 2, pp. 214-238, 1992.

[Walrand, 1988] J. Walrand, *An Introduction to Queueing Networks*, Prentice-Hall, Inc., Englewood, New Jersey, 1988.

[White, 1975] J. A. White, J. W. Schmidt, and G. K. Bennet, *Analysis of Queueing Systems,* Academic Press, New York, 1975.

[Weiselthier, 1993] J. E. Weiselthier, C. M. Barnhart, and A. Ephremides, *Efficient Simulation of DEDS by Means of Standard Clock Techniques: Queueing and Integrated Radio Examples*, Naval Research Laboratory, September 1993.

[Wolter, 1985] K. M. Wolter, *Introduction to Variance Estimation*, Springer- Verlag, New York, 1985.